TECHNICAL SPECIFICATION

VAREK

Version 1.34.0 (current) · October 24, 2026

Deterministic runtime verification of autonomous AI agents using formal methods.

Author: Kenneth Wayne Douglas, MD
Published by Sober Agentic Infrastructure, Inc.
Security contact: kenneth.douglas@soberagents.ai
License: MIT · Project: varek-lang.org · Source: github.com/kwdoug63/varek
Supersedes the v1.12 edition; corrections to earlier editions are folded in and listed in Appendix B.

VAREK is an open-source runtime, with a companion language, for verifying the behavior of autonomous AI agents before that behavior takes effect. Policy is discharged by a satisfiability-modulo-theories (SMT) decision procedure that returns one of three verdicts — SATISFIED, UNSATISFIED, or UNKNOWN — and the runtime is fail-closed: an action proceeds only on an explicit SATISFIED. The stack is vertical, from a declared plan checked before the agent starts to each system call the agent makes at the kernel boundary.

The Warden, VAREK's supervisor, holds each file open, file lookup, outbound connect, program launch and datagram send at the kernel boundary through seccomp-BPF and seccomp user-notify, and decides it before the kernel carries it out. From v1.13 the decision is made in the enforcement path by a purpose-built SMT decision procedure over bounded strings (the object) and 32-bit bitvectors (the open flags), cross-checked against an off-the-shelf SMT solver; from v1.14 it matches exact, prefix, suffix, contains and glob patterns on the resolved canonical path. From v1.15 every authorization carries a certificate that a small, independently written checker must accept before the action takes effect, so the checker, not the procedure, is the trusted decision code for authorizations. From v1.16 the verdict stream is hash-chained, signed with Ed25519 and anchored off the host, so it cannot be rewritten by whoever holds it. From v1.17 the agent runs unprivileged, file lookups are decided like opens, and the Warden's own key, anchor and log are refused by identity through any path. From v1.18 the plan gate, run before the agent starts, runs the cross-action data-flow check, the bounded-refusal breaker and the load-time progress-safety proof, which certifies that an unattended, human-out-of-the-loop (HOOTL) deployment always has an automated next move. From v1.21 the Warden decides each outbound connect on the destination it copied once, dials it itself outside the agent's empty network namespace, and hands the agent the connected socket — the same resolve-then-decide-then-deliver discipline it has applied to files since v1.12. v1.21.1 lets a plan step declare how it opens a

file, so the plan gate can authorize a declared read that the runtime would allow. From v1.24 a policy can name hosts: the Warden resolves every named host itself and serves the agent a hosts view of only those names, so the agent sends no DNS, and each connect is decided on its address and on every name that address belongs to. From v1.25 a policy can opt in to every name under a domain; the agent's lookups then go to a stub resolver the Warden runs, which answers only names a rule allows, within budgets, and records every question. From v1.26 a policy can turn on an egress proxy: connects on the proxied ports go to a separate, unprivileged proxy that reads the name the client asks the server for (the TLS SNI or the HTTP `Host`), and the Warden decides that name, certifies it and dials only an address the name resolves to, so another site on a shared content network is not reached by naming it. From v1.26.1 a policy can have that proxy inspect: it verifies each server, terminates the agent's TLS with a certificate authority made for the run and constrained to the policy's names, and holds each request until the Warden has decided its method, path and query against request rules and the certificate checker has accepted the certificate. From v1.27 a policy can have the agent's program launches after its first decided: at startup the Warden expands the policy's `allow exec` rules to the files they admit and the agent runs in a Landlock domain that lets it launch those files and no others; each launch is decided on its name with a certificate, and before the Warden answers any call of a launched process it checks the process runs a program it decided to run. From v1.27.1, under the same opt-in, every directory the policy lets the agent write is mounted `noexec` in the Warden's own mount namespace, where it opens every file it hands the agent, so nothing written during the run is launched or mapped executable. From v1.28 the agent can hold only placeholders for the credentials it uses: a separate helper fetches each from a file, HashiCorp Vault or AWS Secrets Manager and hands it to the egress proxy, never to the Warden, and the proxy writes it into a request only in the header an allowing rule names, on a request decided SATISFIED; `varek revoke` makes every later call of a run UNSATISFIED. From v1.28.1 every decision record says whom it was for: the user, verified from a token a trusted identity provider signed; the agent, verified from its SPIFFE X.509-SVID; and the sub-agent that made the call, attributed by a control group the Warden gives it at its launch. From v1.29.0 an agent runs with nothing set up first: `varek run` writes a baseline policy for the run with a profile for the agent, the Warden opens and changes files as the agent's own user so the kernel's permissions hold as well as the policy, a dry run decides and records every call while enforcing none but a fixed set of refusals, and `varek policy suggest` drafts a policy from what a run refused. From v1.31.0 a local MCP server the policy declares is confined to the rules written for it, and none of the agent's; a call to a remote MCP server is decided on its method and the tool, resource or prompt it names, read from the body the server will act on; an MCP access token is released only into calls to the server it was issued for, and a token the agent sends that was issued for another is refused; and the verdict stream exports as

OCSF 1.9.0 events a SIEM can read. From v1.32.0 the Warden runs in a container as on a host, from an official, signed image under a published, minimal set of container settings that never includes `--privileged`.

No release from v1.9.1 through v1.34.0 changes the three-state decision semantics or the symmetric-suppression invariant: no extension may move a genuinely unsafe action to SATISFIED.

---

1 Motivation

Agentic AI systems act in the world: they call tools, read and write data, and chain those actions toward goals. The dominant safety posture is probabilistic — a system that is "usually right." Medicine does not deploy systems that are usually right, because the tail is where people get hurt. VAREK applies that standard to agent behavior.

The governing principle is **authorization before execution**: an action is checked against an explicit policy, with a determinate decision, before it is allowed to take effect. The decision procedure does not guess. When it cannot prove an action satisfies policy, it returns UNKNOWN, and the runtime refuses to proceed. A refusal is a safe outcome; an unverified action is not.

---

2 Architecture overview

VAREK is a single vertical stack. Before the agent starts, an optional plan gate checks the agent's declared action-graph — a directed acyclic graph of planned actions — against policy and, with a flow policy, the data moving along its edges. While the agent runs, the Warden holds each policy-relevant system call at the kernel boundary, lifts it into a structured action, decides it, and enforces the verdict before the call completes. System calls the kernel filter admits without asking (memory, time, threads, and reads and writes on descriptors the agent already holds) run undecided; everything else is refused outright.

2.1 Surface language and compilation

Policies are written in the `var::` surface language and compiled to a formal representation suitable for a decision procedure. The compiler is the trust boundary between human-authored intent and machine-checked obligation. The VAREK pipeline language, in which

unsafe operations are not expressible, is stable at v1.0; active development is in the Warden runtime.

2.2 Decision procedure (v1.13, v1.14)

Compiled policy is discharged by an SMT decision procedure returning SATISFIED, UNSATISFIED, or UNKNOWN. UNKNOWN is never coerced to a pass; the runtime treats it as fail-closed.

In the Warden (from v1.13) the procedure is purpose-built for a quantifier-free fragment of two theories: bounded strings for the object an action names (the canonical path, `host:port`, or the exec path) and 32-bit bitvectors for the open flags (`(flags & mask) == value`, faithful to the syscall argument, the kernel's `int flags`, rather than to the descriptor's later state, which `fcntl` can partly change). A policy compiles to one ordered formula per action kind. The same procedure answers concrete per-syscall queries (where satisfiability reduces to evaluation), symbolic-flag queries for the plan gate (SATISFIED only if every admissible flags value is), and rule-reachability queries at policy load (is there any action for which this rule fires first). Policies can say read-only (`readonly`, `access=ro`, `+O_…` / `-O_…`); before v1.13 a path rule admitted every open flag.

From v1.14 a path or exec rule takes a matcher before its constant — `exact`, `prefix` (the default), `suffix`, `contains` or `glob` (`?`, `[...]`, `*`, `**`, and `/**/` for any number of segments) — each a regular language over bytes, matched on the resolved canonical path, so a symlink with an innocent name is decided as its target. A policy can therefore deny a kind of file wherever it appears under an allowed tree: private keys, dotenv files at any depth, a patient's psychotherapy notes. Globs compile to a small automaton stepped word-parallel.

The fragment boundary is a soundness obligation: an over-length string (more than 4,095 bytes), a flag bit outside the ABI set, access mode 3, or a query beyond the enumeration bound is UNKNOWN. Rule reachability is decided exactly over the abstract domain (any byte string up to the bound) by a breadth-first search of the product of the rules' automata, and each reachable rule comes with a shortest witness; at load the Warden reports every rule that can never fire. A general SMT solver is kept off the hot path (v1.5 measured tail latencies of tens of milliseconds for its string theory) and used instead as an independent differential oracle over every repository policy and seeded random policies — zero disagreements over 26,624 checks at v1.13 — joined from v1.14 by a derivative-based procedure that settles bounded reachability where the solver's regular-expression theory does not finish.

Cost. At v1.14 the procedure alone decided a representative path in about 155–165 ns on the finance and healthcare policies. In the live Warden on a 2-vCPU host (`varek bench`,

v1.22.0), the Warden's own time for a decision is 13–17 µs for a refused file open and 56–66 µs for an authorized one, which includes resolving and opening the file and checking its certificate; what the agent waits for the call, notification round trip included, is 53–57 µs and 70–80 µs, against 1–2.4 µs natively (`bench_results_v1_22_0.txt`). From v1.16 a policy's globs are capped at 4,096 tokens, which bounds the work of one decision: on adversarial policies built to hit the worst case, one decision in the live Warden takes about 26 ms (median); real policies are unchanged.

2.3 Certificates: every authorization independently checked (v1.15)

Through v1.14 the Warden authorized an action when the procedure said SATISFIED, so the procedure — about 1,400 lines of C with fast paths, word-parallel automata and a reachability search — was trusted. From v1.15 every SATISFIED verdict carries a certificate naming the deciding rule and a witness that the rule's constant matches, and the Warden authorizes the action only if an independently written checker accepts it. The checker (about 540 lines including SHA-256) has its own policy parser and its own matchers, shares no source with the procedure, and has none of its optimizations. If it refuses, the action is denied and recorded as `certificate_refused`. A test build with a deliberately planted procedure bug shows the effect: the procedure wrongly says SATISFIED for a file under an explicit deny rule, the checker refuses, and the open is denied.

For the property that matters most — no action is authorized that the policy does not allow — the decision procedure moves from *trusted* to *checked*, for logic bugs. The two run in the same process, so a memory-safety bug in the procedure could in principle corrupt the checker's state; this is stated in the trusted-computing-base document. Certificates are recorded in the verdict stream with the SHA-256 of the policy, so `tools/varek_audit.py` can re-check every authorization in a saved run later, from the stream and the policy file alone. Building and checking a certificate adds about 1 µs per authorized open on the example policies.

2.4 Action-graph verification before the agent starts, and the plan gate (v1.6; in the Warden from v1.12.4)

Agent plans are represented as action-graphs and verified compositionally before any constituent action runs. As a plan is revised mid-task it is re-verified, so the authorization guarantee holds across plan changes rather than only at the start.

In the Warden this is the optional `--plan` gate. It decides each `file_open` step on the lexically canonical form of its declared absolute path (from v1.12.4; symlinks are not followed, because there is no agent yet, and every open is still decided again at run time). From

v1.18.0, with `--flow-policy`, it runs the whole v1.7–v1.9 pipeline — the cross-action data-flow check (§2.6), the bounded-refusal breaker and the progress-safety proof (§2.7) — which through v1.17.0 were a library with tests that the Warden did not call. From v1.21.0 it decides `net_connect` steps like the connect they name (§2.14), where through v1.20.0 it refused them all.

Fields on plan steps (v1.20). A step can carry up to 16 `key=value` fields after its target (quoted values may hold spaces, e.g. `contains="a customer record"`), and the flow policy's rules match them like any named argument. Fields are the agent's declarations, like the plan's edges: the node check and the runtime see only the target, and nothing compares the agent's later calls with its fields. Because the agent chooses them, the Warden refuses a flow policy whose rules match a field unless the policy declares `trust_declared_fields`. Targets are limited to 4,095 bytes; the node check calls a longer one UNKNOWN.

Declared open flags (v1.21.1). See §2.15.

Operating and measuring it (v1.22). See §2.16.

2.5 Kernel-boundary integration (v1.7)

The verification layer binds to runtime enforcement at the system boundary. Observed low-level behavior is intercepted, lifted into structured actions, checked against policy, and gated before the action completes. The same policy a developer reasons about is the policy enforced against the agent's actual runtime behavior — not a weaker approximation of it.

2.6 Cross-action data-flow verification (v1.8)

v1.8 extends verification from single actions to data flow across sequences of actions, tracking where data originates and where it is permitted to travel. Policies can express constraints over information movement — for example, that data drawn from a sensitive source must not reach a particular sink — rather than evaluating each action without regard to what came before it. v1.8.0 adds operator-designated, audited declassification (sanitize-then-send), the only mechanism that can bypass the read-secret-then-exfiltrate guarantee, governed by four test-pinned safety properties. From v1.18.0 flow rules can match a URL's host component (`match url.host`), and a plan whose URL a rule cannot read is refused rather than skipped.

2.7 Progress-safety and the bounded-refusal breaker (v1.8.2, v1.9, v1.19)

v1.6–v1.8 prove safety: nothing unauthorized executes. They do not by themselves prove *liveness* — that the system always has a legal, automated next move. An unattended

deployment needs both, or "never requires a human" is a hope: a policy could admit a reachable state in which an action is refused and no authorized fallback exists, a deadlock only a human could break.

v1.9 discharges that obligation once, at policy load, before anything runs. It certifies:

> *For every non-authorizing verdict (UNSATISFIED or UNKNOWN) the policy can produce, the deterministic refusal resolution reaches an automated terminal outcome in finitely many steps, with no point requiring human intervention.*

The proof decomposes into four obligations — bounded refusal, disposed UNKNOWN, disposed exhaustion, and an authorized-fallback reachability proof that composes the underlying decision procedure as its authorization oracle. The result is three-state like every other VAREK verdict: SATISFIED (certified human-out-of-the-loop), UNSATISFIED (a concrete gap, failing obligation named), UNKNOWN (could not decide; fail closed, not certified). From v1.18.0 the Warden runs it on the `--flow-policy` at startup and refuses to start if it fails, so "no human at run time" is provable: if no automated terminal is guaranteed, the system never reaches run time.

The bounded-refusal breaker (v1.8.2) is a loop bound in the trusted boundary, keyed by (session, action-graph signature), so a stuck or adversarial planner cannot resubmit a refused action-graph forever: after the policy's refusal budget, the outcome is a declared terminal (deny, or a pre-authorized action). Each verdict stays a pure function of plan and policy; the breaker only interprets the sequence of verdicts. From v1.18.0 its counts persist across runs in a state file the agent cannot reach. From v1.19.0 a flow policy must also declare `session_refusal_budget N`: the breaker counts every refused plan in a session, whatever the plan, so a planner that changes one step each time is stopped too.

2.8 Enforcement hardening (v1.9.1)

io_uring bypass closed. io_uring dispatches operations from kernel worker threads off the syscall entry path, where a seccomp filter — and therefore the Warden's user-notification mediation — cannot observe them. The Warden's filter refuses io_uring instance creation; there is no sound way to mediate it at this layer. From v1.21.0 `io_uring_setup` answers `ENOSYS` rather than killing the agent, so runtimes such as Node.js fall back to epoll; no ring can be created, and `io_uring_enter` and `io_uring_register` stay in the hard-deny set. The live filter's test (`test_v14_filter`) asserts this.

TOCTOU discipline, measured. When a mediation decision depends on a pointer argument — a path, a socket address — letting the original syscall proceed after approval is unsafe: a

second thread in the target can rewrite the argument between the check and the kernel's use of it, a time-of-check-to-time-of-use (TOCTOU) race. For file opens, the supervisor resolves the approved path itself and injects the resulting descriptor into the target, so the target's syscall never runs against mutable memory. On the race harness (`tests/seccomp_toctou_harness.c`), approve-then-continue leaked the protected target 1,848 to 1,889 times in 20,000 attempts and resolve-and-inject leaked 0 (three runs each on a 2-vCPU host, `tests/toctou_results_v1.18.0.txt`; the count depends on the host). Connects, for which there is no file descriptor to resolve, were deny-only from v1.9.1 through v1.20.0; v1.21.0 applies the same discipline to them (§2.14). Agent-initiated `execve` remains deny-only.

2.9 Mediation completeness (v1.9.2)

v1.9.2 inverts the Warden's enforcement model and wires the result into the live supervisor. The filter is *default-deny*, so the residue — every syscall not explicitly admitted — is refused by construction rather than by enumeration. Completeness of mediation is treated as a closure property: for each policy-relevant effect, the obligation is to have enumerated all the syscalls that produce it, across all application binary interfaces.

Default-deny allowlist. Unknown syscalls, variant syscalls (`clone3`, `openat2`, `faccessat2`, the `pidfd_*` family), and the 32-bit multiplexers (`socketcall`, `ipc`) are denied unless admitted — a bypass that was not enumerated becomes one that was not admitted.

Native-ABI lockdown. The deny default applies across every architecture, and no secondary ABI is admitted, so the same operation re-issued through the 32-bit compat ABI (`int 0x80`) or the x32 ABI (`__X32_SYSCALL_BIT` set) hits the deny path — closing the most common real-world seccomp escape, asserted on the live kernel by a release-blocking test.

Hard-deny set and namespace denial. Syscalls with no legitimate use inside a mediated agent — `ptrace`, `bpf`, `userfaultfd`, `process_vm_readv`/`writev`, `pidfd_getfd`, the mount/FUSE family, the kernel-module and `kexec` family, `perf_event_open`, and the key-management calls — are denied with process termination in strict mode. `clone` and `unshare` are filtered on their scalar flags argument (a register value, so the check is race-free) to deny `CLONE_NEWUSER` and the namespace-creation set. `clone3`, whose flags live behind a pointer, never executes: from v1.12.2 it is answered `ENOSYS`, so the C library falls back to `clone` and every thread or child passes the same flag filter.

Integration and validation. The default-deny baseline replaces the filter the Warden installs. A conformance target — a running, agent-shaped workload — exercises the boundary end to end: it opens an allowed file (served by descriptor injection), is refused a denied path, creates a socket, and is refused an outbound connection (deny-only through v1.20.0; from

v1.21.0 refused because the conformance policy has no host rule allowing it). Under live enforcement it passes every phase. An observe mode logs rather than blocks, so a target's required syscalls can be harvested before the deny default is turned on, while the hard-deny set still terminates on a dangerous call. The conformance target also surfaced, and verified the fix for, a regression in which deny-only exec mediation refused the target's own bootstrap `execve`; the Warden authorizes exactly the operator-specified launch, once per run.

2.10 Supervisor/target lifecycle coupling (v1.9.3)

Enforcement assumes the supervisor is alive. If it stops, mediated calls fail closed, but an agent that keeps running can still act through capabilities it already holds. v1.9.3 removes that window in the live Warden.

The target requests a kill signal on supervisor death before it installs its filter, and first confirms, through a pipe only the supervisor holds open, that the supervisor is still alive. It runs as the first process of a dedicated PID namespace, so when it dies the kernel kills every other process in the namespace; on orderly shutdown the supervisor kills the whole tree. The supervisor watches the target through a process descriptor. The namespace requires `CAP_SYS_ADMIN`; the Warden checks for it at startup and refuses to run without it unless explicitly told otherwise. A crash test kills a live supervisor with `SIGKILL` and, separately, stops it with `SIGTERM`, and asserts no agent process survives either; against the pre-fix Warden it fails (two survivors after a crash, one after orderly shutdown).

2.11 Mediation correctness (v1.12.0–v1.12.4)

A decision is only as sound as the identity of the object it is made about, and a record is only evidence if the process it describes cannot forge it.

Resolve-then-decide. Through v1.9.3 the Warden matched policy against the pathname string the agent supplied and then, separately, opened whatever that string resolved to — two steps that could name two different objects: `/allowed/../../denied` matched an allowed prefix as text, a symlink planted inside an allowed directory was followed to a denied target, and `/proc/self`, which the kernel resolves in the caller's context, resolved to the supervisor rather than the agent. From v1.12 the Warden resolves the object once, reads the canonical path back from the resolved descriptor, decides policy on that path, and hands the agent the same descriptor, so decision and delivered capability refer to the same inode by construction. From v1.12.1 it pins the object with `O_PATH`, decides, and only after an ALLOW opens it with the agent's flags, so a denied open has no side effect (no truncation, no creation, no FIFO that wedges the supervisor). From v1.12.3 ordinary symlinks are followed and decided as their targets, so dynamically linked agents (CPython, a JVM, Node.js) can load

their libraries; a leading `/proc/self` or `/proc/thread-self` is mapped to the agent's own process, and any other process's `/proc` is refused after resolution. Because the decision is on the canonical path, `allow path` prefixes name canonical locations (`/usr/lib/`, not `/lib/`).

Record integrity. Every agent-controlled field is escaped, so no agent string can begin, end or fabricate a record, and each record reports the resolved canonical object it was decided on. From v1.12.1 the agent's own stderr is relayed with an `[agent]` prefix and every record carries a per-run id and a contiguous sequence number, so a forged, missing or foreign record is detected.

Threads, children and inbound networking. From v1.12.1 `bind`, `listen` and `accept` are refused and the agent runs in its own network namespace, holding only a loopback interface. From v1.12.2 agents can start threads and wait for their children, and the launch exec is granted once per run, not once per process, closing a race in which a thread could change the launch path.

2.12 Lookups, protected objects and an unprivileged agent (v1.17)

Lookups are decided. `stat` (`newfstatat`), `statx`, `access`, `faccessat`, `faccessat2`, `readlink` and `readlinkat` were admitted and unlogged, so an agent could learn whether any file existed and read where any link pointed. From v1.17 the Warden resolves the name like an open, decides it with the SMT decision procedure as a read-only open of the same object (`access(W_OK)` as a write), certifies an ALLOW, records it, and answers the call itself. A name that does not exist is decided on where it would be — `ENOENT` inside the policy, `EACCES` outside it — so a lookup says nothing about what exists elsewhere. Lookups of the directories leading to an allowed path are answered with existence and type only, and recorded as `metadata_ancestor`; lookups of descriptors the agent already holds are answered like `read()`.

Protected objects, refused by identity. Every open and lookup is checked after resolution, on the object actually pinned and before the policy is asked, against the device and inode of the signing key, the anchor and the verdict stream, so a bind mount, overlay or hard link into an allowed tree cannot reach them (`protected_object`). Raw storage and memory devices — block devices, `/dev/mem`, `/dev/kmem`, `/dev/port`, the sg, bsg and NVMe command devices, `/proc/kcore` — are refused the same way whatever the policy says (`raw_device`).

The agent runs unprivileged. Through v1.16.3 the agent ran as root with every capability, held back only by the filter. From v1.17 it runs as `nobody` or the user `--run-as` names; before the filter is loaded, the Warden's child clears its supplementary groups, empties the capability bounding and ambient sets, switches group and user, and verifies that no capability remains. File access does not depend on the agent's rights: the Warden opens allowed files on its behalf only after the policy allows it.

2.13 A verdict stream its holder cannot rewrite (v1.16, v1.18)

From v1.12.1 the supervised agent cannot forge, drop or reorder a record. From v1.16 neither can a person who holds the log afterwards. Every record is hash-chained; with `--sign-key`, run start, a checkpoint every 64 records (and at least once a second) and run end are signed with Ed25519; with `--anchor`, each checkpoint is appended to storage the log's holder cannot rewrite. `tools/varek_audit.py --pubkey --anchor` catches any edit, insertion, removal or reordering before the last signature, and a stream cut short, by anyone without the key, and any rewrite of anchored history even by someone with it. It verifies signatures with its own pure-Python RFC 8032 verifier. The Warden refuses to start if the policy would let the agent open the key, the anchor, the verdict stream or a raw disk.

An anchor on the Warden's own host does not protect against that host's root, who also holds the key. From v1.16.2 `tools/varek_anchor_forward.py` sends each checkpoint off the host, normally within a second, spooling through outages, to an append-only receiver set up by `tools/varek_anchor_receiver.sh` — an SSH account restricted to one forced command that appends well-formed anchor lines to a `chattr +a` file. `tools/varek_preflight.sh` (v1.16.1) checks a deployment ahead of its first run, including every location the Warden refuses at startup, and with `--run` performs an audited trial run.

Evidence export. The verdict stream exports as a Bill of Materials in the CycloneDX 1.6 format: the runtime as a tool component, the supervised run as the subject, each authorized object and connected destination as a component, and an Authorization-Before-Execution attestation. From v1.18.0 the attestation is derived from the stream, the exporter checks the policy file against the recorded SHA-256, the BOM can be signed (JSON Signature Format, Ed25519), and output is tested against the published CycloneDX 1.6 schema. The exporter refuses a stream that does not verify, so a corrupted log cannot be laundered into an attestation. (CycloneDX is a trademark of the OWASP Foundation; VAREK is not affiliated with or endorsed by the OWASP Foundation or the CycloneDX project, and uses the name only to describe interoperability with the openly published CycloneDX format, standardized as ECMA-424.)

2.14 Decided connections (v1.21.0)

Through v1.20.0 a supervised agent had no network: every connect was refused, whatever the policy said, because letting an allowed connect continue in the kernel would let a second thread change the destination between the Warden's check and the kernel's read of it. From v1.21.0 the Warden decides each outbound connect and carries it out itself:

1. The destination is copied once from the agent's memory.

2. It is spelt canonically: `a.b.c.d:port`, `[IPv6]:port` (an IPv4-mapped address as its IPv4 form), or `unix:<canonical path>` for a Unix socket named by a path, resolved like a file open.
3. The SMT decision procedure decides it against the policy's `host` rules, and the independent checker must accept the certificate of any ALLOW.
4. The Warden, in the host's network namespace, makes a socket of the agent's kind, copies over the socket options the agent set (from a list of 58), and connects it to the copy it decided on; a Unix connect is made with the agent's uid and gid.
5. `SECCOMP_IOCTL_NOTIF_ADDFD` puts the connected socket in place of the agent's descriptor, with the same number and the agent's close-on-exec and non-blocking state, and the agent's `connect` returns what its own would have.

The kernel never reads the agent's socket address, so the race stays closed: in 2,000 attempts to swap the destination from an allowed one to a denied one after the check, the denied side was reached 0 times. The agent's own network namespace stays empty; the only way out is a socket the Warden made. Covered: TCP and connected UDP over IPv4 and IPv6, and Unix stream, datagram and seqpacket sockets named by a path (IPv6 decisions are tested; IPv6 dialing was not exercised on the release host).

A send that names a destination is still refused, because an unconnected datagram cannot be decided without a race on its address; a send with no destination goes to the peer of a connect the Warden decided. Routing options that could redirect a connected socket's packets (`IP_OPTIONS`, `IPV6_RTHDR` and two legacy IPv6 forms) are refused on every socket. `bind` is refused except the wildcard, port-0 form libuv uses before connecting a UDP socket; `listen` and `accept` stay refused.

Cost. On a 2-vCPU host a connect took about 75 to 125 µs longer than a native one at the median; a whole Python `requests.get` to a local server took about 0.17 ms longer (1.21 → 1.38 ms). A profile put about 2 µs of the Warden's part in the decision and certificate check.

Scope. VAREK decides *where* an agent may connect, not what it sends: an allowed host is a channel. Through v1.23 rules matched only the numeric address dialed; from v1.24.0 they can name hosts (§2.18), and from v1.26.0 connects on proxied ports are decided on the name sent to the server (§2.20).

2.15 Declared open flags at the plan gate (v1.21.1)

A plan step has no open flags of its own, so through v1.21.0 the plan gate decided a `file_open` step with the flags unknown. A path allowed only by a rule with a flag clause (`readonly`, `access=ro`, `-O_TRUNC`) was therefore UNKNOWN at the gate, although the runtime

allowed the same open — and all five shipped sector policies grant their libraries read-only. From v1.21.1 a `file_open` step may declare how it opens its file:

```
action load  file_open  /srv/agent/reference/in.json  open=read
action save  file_open  /srv/agent/work/out.json      open=O_WRONLY|O_CREAT|O_TRUN
edge   load  save
```

`open=read` means `O_RDONLY` and nothing else; otherwise the value is one access mode followed by any of 17 named flags, each at most once, joined by `|`. A name has the value an agent's `open()` passes for it (`O_LARGEFILE` is the kernel's bit, as in the policy language). Anything else — another word, a lowercase name, a repeated flag, an `open` field on a step that is not a `file_open` — makes the step UNKNOWN, never SATISFIED, with the reason logged. The node check decides and certifies the step with exactly the declared flags, as the runtime decides an open with the flags the agent passes. The field is a declaration: it does not bind the agent, and an open with other flags is decided on its own flags at run time.

2.16 Operating and measuring it (v1.22)

Through v1.21 the Warden and its tools each took their own arguments: the policy, the verdict stream, the signing key, the anchor and the agent's user for the Warden, then the policy checker, the audit and the CycloneDX exporter afterwards. v1.22 puts them behind one command, `varek`, which reads the shared settings from `/etc/varek/varek.conf`: `doctor` checks the host, `init` writes the settings and the signing key, `run` supervises an agent, `refusals` explains each refusal by the policy line or the reason that produced it, `audit` re-checks a run's certificates, chain and signatures, and `export` writes signed CycloneDX 1.6 evidence. It wraps the existing tools and changes no verdict semantics; `--show-commands` prints each underlying call.

`varek bench` measures what mediation costs on the host it runs on. A fixed workload runs natively and as the agent under the Warden, alternating, for six kinds of call (an allowed, a denied and an unmatched file open; an allowed and a denied connect; a small whole request), and reports per kind what the agent waited, what the Warden added and the Warden's own decision time from its records. It checks every verdict while it measures: each allowed call certified and successful, each refused call `EACCES`, and the listener behind a denied destination never reached. It decides with the active policy's rules followed by its own, so the cost of a real policy is in the numbers. The figures in §2 come from it.

2.17 VAREK Enterprise on AWS Marketplace (v1.23)

v1.23.0 is the first VAREK Enterprise image on AWS Marketplace: an Amazon Machine Image (AMI) on Amazon Linux 2023 (x86_64) with the Warden, the `varek` command, the Enterprise policy packs (HIPAA, SOC 2) and an AWS License Manager entitlement check. The image is built from the packaging in the source (`varek/v1_4/packaging/aws-marketplace/`): the build installs the runtime, removes the compiler, proves the installed image with a real run, audit and signed export, and hardens it to the AWS Marketplace AMI policy (key-only SSH, no root login, no passwords or stored keys). On the image, choosing an Enterprise pack checks the buyer's contract with License Manager; without an entitlement `varek` explains and lists the VAREK Core packs. The license gates only the Enterprise packs: the Warden and the Core packs never depend on it, and off the image nothing is checked. The Warden's decisions are unchanged from v1.22.0; the only code change is the version it reports, so the release, the image and the listing agree. Deploying, operating, backing up, upgrading and supporting the image are covered in the deployment guide (`docs/aws-deployment-guide.md`).

v1.23.1 fixes the license check, which on the v1.23.0 image refused every subscriber. AWS Marketplace issues each contract dimension as a License Manager Count entitlement; the check now asks for one unit and returns it at once, so the next check is not refused while a checkout holds it. The fix was found before any customer was affected, by an end-to-end test through a private offer to a second AWS account. v1.23.1 also lets the VAREK Core packs read `/usr/share/` (time zones, locale) and points the anchor forwarder's service unit at the installed tools. The Warden's decisions are unchanged.

2.18 Host names without agent DNS (v1.24.0)

Through v1.23 a host rule matched only a numeric address. A policy either hard-coded addresses, which break when a cloud provider or content network moves them, or let the agent reach a DNS server, through which it can send data out in the questions it asks. From v1.24.0 a policy names hosts:

```
require warden 1.24
allow host api.salesforce.com:443
```

1. **Grammar.** A name is lowercase A-labels with no trailing dot, labels of at most 63 bytes and 253 bytes in all; a name without a port matches every port. Wildcards are refused (planned for v1.25). Names take effect only after `require warden 1.24`, so an older Warden refuses the policy instead of loading rules it cannot match. The SMT decision procedure, the certificate checker and the cross-check oracle parse names independently and must agree rule by rule.

2. **Resolution table.** Before the agent starts, the Warden resolves every name a host rule names, allow or deny, A and AAAA, through the host's resolver. It refreshes each at its TTL, clamped to [30 s, 1 h]; an address that leaves an answer stays valid for its last TTL, at most 5 minutes. Lookups run in a resolver helper process, so the process holding the signing key never parses network data and never waits on DNS while an agent thread waits on it. Each lookup is a chained `resolution` record.
3. **Views.** The agent reads `/etc/hosts`, `/etc/resolv.conf`, `/etc/nsswitch.conf` and `/etc/host.conf` from sealed memfds the Warden writes. Its hosts view lists localhost and the allowed names' current addresses, IPv4 first, and nothing else; its resolver configuration names a server no one answers. Every connect to port 53 is refused, and a writable open of any of the four paths is refused, whatever the policy says.
4. **Decisions over every name of an address.** A connect is decided on its address and on `name:port` for every name the address belongs to, current or in grace. The first rule, in policy order, that holds on any of them decides, so a deny on a name holds on its addresses. The certificate covers the deciding name, and the checker confirms that no earlier rule holds on another candidate. A loopback, link-local, unspecified or multicast address is decided on the address alone, so a name's DNS cannot lead the agent to the host's own services. The record carries the address dialed, every candidate and the table generation.
5. **Audit.** `varek_audit.py` checks that each candidate was bound to the address dialed by the resolution records, that no name of that address was left out, that each connect names the latest table generation, and that addresses and candidates are spelt as the Warden spells them. A stream edited to drop a name fails, even with its hash chain recomputed.

The plan gate decides `net_connect api.example.com:443` steps on the name.

Measured. The Warden fetched three URLs by name once a minute for 24 hours on a cloud host, one each from Fastly, Cloudflare and CloudFront: 4,320 of 4,320 fetches succeeded while CloudFront changed its answer 2,034 times, no connect was refused, and the audit passed on the 114,279-record stream. Deciding a connect over 16 candidate names cost no time that could be measured end to end; the decision procedure takes about 0.5 µs per candidate.

Scope. A name decides which addresses the agent may reach, not which site it asks for there: an allowed name on a shared content network also reaches the other sites served from its addresses, by another name in TLS SNI or the HTTP `Host` header. The Warden trusts the host's resolver, or the one named by `--dns-server`, and does not validate DNSSEC itself, so whoever controls an allowed name's DNS chooses its addresses, private ones included.

Deciding on the name sent to the server is the v1.26 egress proxy (§2.20). Wildcard names are §2.19.

2.19 Wildcard host names, opt-in (v1.25.0)

v1.24 needs every name listed in advance. An agent that reaches per-tenant names (`acme.api.example.com`, `globex.api.example.com`) cannot list them. From v1.25.0 a policy can allow every name under a domain:

```
require warden 1.25
allow host *.example.com:443 acknowledge=dns-channel
```

Those names are known only when the agent asks for one, so under a wildcard the agent's lookups have to leave the host. The design keeps that path narrow, visible and bounded:

1. **Grammar.** `*.` is the whole leftmost label, over a suffix of at least two labels that follows the v1.24 name rules; `*.example.com` matches `a.example.com` and `a.b.example.com`, never `example.com`. Each of the three parsers converts the rule to a glob (`?*.example.com:443`), so it is decided, certified, checked for reachability and fuzzed like any glob, with no new matcher.
2. **The acknowledgment.** Every wildcard allow rule must carry `acknowledge=dns-channel`, or the policy is refused, so the policy a reviewer reads states that the agent's lookups under the rule leave the host.
3. **Shared domains refused at load.** A wildcard over a domain where anyone can create a name would allow an attacker's endpoint. It is refused, naming the entry, when its suffix is a public suffix (`*.com`), is or lies under an entry of the Public Suffix List's private section (`*.s3.amazonaws.com`, `*.myorg.github.io`), or is on a VAREK list of multi-tenant domains the Public Suffix List lacks (`*.my.salesforce.com`, `*.slack.com`), or lies above any of these (`*.salesforce.com` would cover `my.salesforce.com`). Both lists ship pinned: the Warden refuses lists in its data directory that are not the release's, and records their SHA-256 and whether they are in `run_start`.
4. **The stub resolver.** The agent's resolver configuration names `127.53.53.53:53`, a UDP and TCP resolver the Warden runs inside the agent's own network namespace. A name no allow rule can reach is answered NXDOMAIN, and no question leaves the host. A name a wildcard allows is looked up by the Warden's resolver helper when the agent asks, never on its own, and recorded. Every other connect to port 53 is refused.
5. **Budgets.** Each wildcard allow rule has a budget of new names a run (`names=`, default 256), of lookups sent upstream a minute (`rate=`, default 30, a name asked again after its

TTL included), and at most 63 bytes before the suffix. A question past a budget is answered NXDOMAIN and not looked up. A new name carries at most about 41 bytes the agent chooses, so at the defaults new names carry at most about 10.5 KB a rule a run; after that, the choice of names asked again carries at most about 30 bytes a minute.

6. **Records and audit.** Every question to the stub is a chained `dns_question` record. `varek_audit.py` checks the budgets against the policy file, every charge against them, and that every name looked up was asked for.
7. **Many names on one address.** Per-tenant names behind a CDN share addresses. A connect is decided over every name its address belongs to, with no limit; past 15 names the record carries their count and SHA-256, and the audit rebuilds them from the resolution records.

Measured. A name already in the table is answered by the stub in about 50 µs; a new name costs the two upstream questions (A and AAAA) plus 0.7 to 1.2 ms for the hand-off to the resolver helper and the records. A connect decided over 41 candidates on one address, hashed for the record, takes the Warden 34 to 79 µs more at the median than one allowed by a numeric rule, about 1 to 2 µs a name. A 24-hour run against 40 Wikipedia language editions served from one address fetched 1,440 of 1,440 times with no refused connect, every one of its 2,880 questions recorded, and no budget refusal; it used 40 of 256 names and at most 2 a minute of 30.

Scope. The name channel is bounded, not closed: within the budgets, labels the agent chooses, and which names it asks again and when, reach the host's resolver and the domain's DNS servers, which a DNS provider rather than the API's owner may run. The lists are snapshots; a domain where anyone can create a name and which neither list holds is not refused. With the v1.26 egress proxy on, names the agent asks for on proxied ports get synthetic addresses and are not looked up; the Warden looks a wildcard's name up only to dial a request the policy allowed, charged to the same budgets (§2.20).

2.20 The egress proxy, SNI mode (v1.26.0)

v1.24 and v1.25 decide a connect on the address dialed and the names that address belongs to. On a shared content network that is not enough: every site served from the address shares its names' fate, and a client can ask the server for another site while the Warden's decision holds. From v1.26.0 a policy can turn on the egress proxy:

```
require warden 1.26
proxy on
allow host api.example.com:443
allow host *.svc.example.com:443 acknowledge=dns-channel
```

The Warden then decides connects on the proxied ports (`proxy ports`, default 80 and 443) on the name the client sends, not on the address. A separate proxy reads the name; the Warden decides, records and dials; the proxy relays.

1. **Grammar.** `proxy on`, `proxy ports P...` (at most 16) and `proxy upstream http://HOST:PORT`, after `require warden 1.26`. The SMT decision procedure, the certificate checker and the cross-check oracle parse them independently, and the Warden refuses to start if its two C parsers read them differently. `proxy inspect` is refused until v1.26.1.
2. **The proxy process.** A mode of the Warden's own binary, started before the agent as its own user (never root, never the agent's user), with no capabilities, no-new-privileges, and only its control socket and its listener open. It proves its user to the Warden with `SCM_CREDENTIALS`, listens on 127.0.0.1 in the host's network namespace, which the agent cannot reach, and exits with the Warden. The Warden refuses a `proxy on` policy it cannot start the proxy for, rather than run it without one.
3. **Synthetic addresses.** A name that host rules allow only on proxied ports is never resolved for the agent: the hosts view and the stub give it a stable address from 198.18.0.0/15, recorded once. Nothing is looked up and no wildcard budget is charged when the agent asks.
4. **The hand-off.** A TCP connect on a proxied port is handed to the proxy unless a numeric rule allows the address (dialed directly, as in v1.21) or a rule denies it (refused); a synthetic address is always handed over. The Warden binds a socket to 127.0.0.1, announces its port to the proxy, connects it to the listener and gives the agent that socket. The proxy closes any connection the Warden did not announce. A UDP connect on a proxied port that only a name allows is refused, so QUIC clients fall back to TCP.
5. **What the proxy reads.** Bounded parsers that allocate nothing read a TLS ClientHello's SNI (at most 16 KB over 64 records), an HTTP/1.x request's one `Host` (headers within 8 KB, CRLF only), or a `CONNECT host:port` whose ClientHello must carry the same name. They refuse what they cannot read exactly: no SNI, two names, an IP literal, Encrypted Client Hello and ESNI, a repeated extension, a `Host` port other than the one connected to. Only a connection id, the kind, the name and the port reach the Warden.

6. **The decision.** The Warden decides `name:port` with the SMT decision procedure, and the certificate checker must accept the certificate. It takes the name's addresses from the resolution table (a wildcard's name is looked up on demand, charged to the rule's budgets), skips special addresses (loopback, link-local, unspecified, multicast, cloud metadata, IPv4-mapped and IPv4-translated IPv6), synthetic addresses and addresses a rule denies, dials without blocking, and passes the connected socket to the proxy over `SCM_RIGHTS`. The Warden keeps each hand-off's state: one report each, on the port the agent connected to; anything else from the proxy stops the run. Plain HTTP is read request by request: each must name the host and port decided, and each body passes as its framing says.
7. **Records and audit.** Each hand-off, decision (`net.proxy`) and close (`proxy_close`, with the proxy's byte counts) is a chained record; a connection the proxy never reported is recorded `unreported` at run end, and if the proxy dies the run stops. `varek_audit.py` asks each hand-off and each dialed address of the policy again, checks one certified decision per hand-off, a close for each connection passed on, and that no connect on a proxied port was dialed directly except by a numeric rule.
8. **Chaining to a customer's proxy.** With `proxy upstream`, the proxy asks the upstream with `CONNECT name:port` after the Warden decides, so both apply and a refused name never reaches the upstream; the upstream's own refusal is recorded with its status.

Measured. Over 1,000 new connections each on a 4-vCPU cloud host, the proxy adds about 1.0 ms at the median to a TLS request (full handshake) over a direct dial, and 0.35 ms to a plain HTTP request; deciding, certifying and dialing take the Warden 105 µs at the median (p99 234 µs). A 24-hour run against three content networks is reported in the release notes.

Scope. Inside TLS, SNI mode reads only the ClientHello: a request inside TLS can carry a `Host` for another site on the same content network (domain fronting), and later requests on a kept-open TLS connection can name another host, though both reach only the server the allowed name resolved to. The gap is closed on proxied ports only. With an upstream, the address checks are the upstream's, since the Warden never sees the address it dials. The close records' byte counts are the proxy's claims. Rules on method, path and query, deciding every request inside TLS, are the v1.26.1 inspecting mode (§2.21).

2.21 The egress proxy's inspecting mode (v1.26.1)

SNI mode decides which site a connection asks for, but not what it asks there: once a name is allowed, any request on the connection passes, including one whose `Host` names another site on the same content network. From v1.26.1 a policy can have every request decided:

```
require warden 1.26
proxy inspect
allow host api.example.com:443
deny  request * https://api.example.com/v1/admin/**
allow request GET https://api.example.com/v1/models
allow request GET https://api.example.com/v1/files?limit=*
allow request POST https://api.example.com/v1/chat/completions max_body=256k
```

A connection is decided on its name as in §2.20; then each request on it is decided before any of it reaches the server.

1. **The object.** A request is decided on `METHOD scheme://name:port/path?query`: the name and port are the connection's, decided already (the request's `Host` must equal them), and the method, path and query are the request's own bytes. A request rule is held as a glob over that object (a `*` method is any method, a `*.suffix` host a wildcard, the port filled in, `*`, `**` and `[...]` in the path glob wildcards, `?` the query's literal `?`), so it is decided, certified, checked for reachability and fuzzed with the string fragment of §2.2, with no new matcher. Request rules are first-match, and a request no request rule allows is refused: in inspecting mode a host rule lets the agent connect, and only request rules let it ask.
2. **The query.** A request, and a rule, has at most one `?`, so a rule's `?` meets only the request's and its path wildcards only the path. An allow rule without a `?` holds on no request with a query, not even an empty one; a deny rule without one holds on its path with any query. The three parsers (the decision procedure, the certificate checker and the cross-check oracle, which also states it to its SMT solver) define "holds" this way, so decisions, certificates and reachability agree.
3. **Read one way.** A server and the matcher must not read a target differently, so the proxy refuses rather than normalizes one some server would read as another path: `.` and `..` segments, `//`, `\`, `;`, `#`, bytes outside 0x21–0x7e, percent-encoded `/`, `\` or unreserved bytes, bad escapes, a second `?`, and any form but origin-form (absolute-form only for the connection's own scheme and authority), with ambiguous framing and protocol upgrades. The Warden checks the object's form again itself before deciding. A rule's own path is held to the same forms.
4. **Terminating TLS.** The proxy, now its own binary (`warden-proxy`, copied into a sealed memfd, hashed into `run_start` and started from the copy with `fexecve`, so what runs is what was hashed), first verifies the server: TLS 1.2 or later, the name decided checked in the certificate's subject alternative name, against the host's trust bundle or one the operator names. Then it answers the agent with a leaf for the name from a certificate authority made for the run: ECDSA P-256, valid seven days, its key in OpenSSL's secure

heap where the memory-lock limit allows and the proxy not dumpable, with critical name constraints that permit only the names the policy lets the agent reach on proxied ports and exclude passthrough hosts and every IP address. The Warden serves the agent the host's bundle with that CA added at the system bundle paths, the CA and a PKCS#12 store at fixed paths, and names them in the agent's environment, so common clients (Python, curl, Node.js, Java) trust it without changes. ALPN offers `http/1.1` only; no session tickets.

5. **The gate.** The proxy reads each request's head (8 KB at most), reports its object, and lets nothing of it through until the Warden's verdict; pipelined and kept-alive requests are each decided in turn. The Warden decides the object with the SMT decision procedure, the certificate checker must accept the certificate, and a body declared over the allowing rule's `max_body` is refused. On a refusal nothing more of the connection reaches the server, and the agent gets a 403 with its reason.
6. **Bodies.** A chunked body is sent up to `max_body` and the connection cut there. Each allowed body's length and SHA-256, as sent, are recorded (marked incomplete if the connection ended first); bodies are never stored.
7. **Passthrough hosts.** `proxy passthrough host NAME` keeps a host whose client pins its certificate in SNI mode, decided on its name and recorded as not inspected.
8. **Records and audit.** Each request is a chained `net.request` record with its decision, rule line and certificate, each body a `request_body`, and each inspected connection's close counts its requests; `run_start` records the CA's, trust store's and `warden-proxy`'s SHA-256. `varek_audit.py` checks that each request belongs to a connection passed on to be inspected, in order and before its close, in a form the proxy must accept; re-checks each allowed request's certificate and asks each refused one of the policy again; and checks each body against its rule and declared length, each close's count, and the trust record against the policy. A misbehaving proxy (a request out of order, a body unreported, an object in a refused form, a wrong count) stops the run.

Measured. Over 1,000 requests each on a 4-vCPU cloud host, inspecting mode adds about 0.8 ms at the median to a new TLS connection over SNI mode (the agent's handshake with the run's CA) and about 0.2 ms to a request on a kept-alive connection; the Warden decides and certifies a request in 3 to 13 µs at the median. Measuring it found and fixed a 40 ms stall on each new connection: with no session tickets the proxy sent nothing after the client's Finished, so the client's next small write waited for a delayed acknowledgement; the proxy now acknowledges at once during the handshake. A 24-hour run against three content networks in inspecting mode fetched 1,296 of 1,296 times with no allowed request refused, refused and recorded all 144 probes (fronting by SNI and by `Host`, a path written another

way, a denied path and an unlisted one), and passed the audit on the 19,658-record stream; the Warden decided and certified a request in 15 to 18 µs at the median.

Scope. What is decided is the request line: headers are not decided (beyond `Host` and framing) and bodies only by length, so what the agent puts in an allowed request reaches the server the policy allows. The proxy sees plaintext and holds the run's CA key. The path and query are recorded as sent, so a secret in a URL is in the stream. Clients that bring their own trust (a pinned certificate, an application-bundled store) fail unless their host is a passthrough host, and passthrough hosts are not inspected. HTTP/1.1 only. A server any root in the trust bundle vouches for is believed.

2.22 Decided program launches (v1.27.0)

Through v1.26.1 the agent's own launch is allowed once and every later `execve` or `execveat` is refused, whatever `allow exec` says: answering an allowed launch with `SECCOMP_USER_NOTIF_FLAG_CONTINUE` lets the kernel read the path again from the agent's memory after the decision, and another thread can swap it in between. Opens and connects avoid this because the Warden performs them and hands over a descriptor; a launch must replace the agent's own image, so the kernel has to perform it. From v1.27.0 a policy can opt in:

```
require warden 1.27
allow exec /usr/bin/git
allow exec /usr/bin/python3
allow exec prefix /usr/libexec/gcc/x86_64-linux-gnu/13/
```

1. **The opt-in.** `require warden 1.27` turns launches on in the decision procedure, the certificate checker and the cross-check oracle; the Warden refuses to start if the procedure and the checker read it differently, and a launch is allowed only where the checker's own reading turns launches on. Without it a policy behaves as on v1.26.1.
2. **The launch set.** At startup each `allow exec` rule is expanded to the files it admits (an exact path, or a prefix or glob by a bounded walk), each candidate decided by the decision procedure, an earlier deny winning and a deny on a file's own path keeping it out whatever symlink reaches it; only regular files with an execute bit, at most 256 a rule. Each dynamic program's loader (its `PT_INTERP`, read with a bounded ELF reader, not followed further), the operator's program and a script's `#!` interpreter are added. The set is held by device and inode, with each file's canonical path and SHA-256, recorded in `run_start`. A rule that cannot be expanded to files refuses the start.

3. **Landlock.** The agent runs in a Landlock domain that handles `LANDLOCK_ACCESS_FS_EXECUTE` and grants it on those files only, applied after it drops its privileges and before seccomp and its own launch, and inherited by everything it starts. The kernel then refuses to launch any other file: one written during the run, an `O_TMPFILE` or unlinked file through `/proc/self/fd` or `execveat`, a file in `/dev/shm` (measured, §5). Landlock does not govern a memfd; the agent cannot make one (`memfd_create` is in the hard-deny set, kept there by a build check). Without Landlock every launch stays refused.
4. **Deciding.** A launch is decided on its name: the path as written made absolute, `.` and `..` resolved, a final symlink keeping its name, `execveat(fd, "", AT_EMPTY_PATH)` on the descriptor's file; by the SMT decision procedure, with a certificate the checker must accept. An allowed launch must name a file of the set (same device and inode) and is answered with CONTINUE; a launch the policy allows of a file not in the set is refused.
5. **The identity check.** A swap after the decision can reach only a file in the set. Before the Warden answers any call of a process that has launched, `/proc/<pid>/exe` must be a program it decided to run; the dynamic loader, in the set only because the kernel needs it, is never one. A process that is not is killed before its call is answered, and the run fails. Each allowed launch gets one `exec_result` (`launched`, `failed`, `gone`, `killed`); two threads launching at once resolve to one launch and one failure.
6. **What a compile needs.** With the opt-in, `access(X_OK)` answers whether a launch of that name would be allowed, a lookup under a missing directory is walked as the kernel walks it, and an open of a missing file the policy would let the agent read answers ENOENT, certified. gcc compiles and links under the Warden with each of its launches decided.
7. **Audit.** `varek_audit.py` asks the policy again about the set, each launch and each refusal, requires one `exec_result` per allowed launch, holds each refusal to where the Warden can write it, and fails a run in which a process was killed.

Measured. An allowed launch costs about 0.3 ms more than natively at the median on a 4-vCPU host; the Warden decides it in 72 µs (a refused one in 46 µs), and the identity check is within the spread between runs. A 24-hour run on a 1-vCPU, 1 GB host launched `git`, `python3` and a compiler a minute in turn: 1,296 tasks all succeeded, 3,456 launches allowed and each seen running, 144 probes (an unlisted program, a denied one, one written during the run) all refused and recorded, no process killed, and the audit passed on the 282,662-record stream; a 6-minute run of the reviewed build, a task every 5 seconds, passed the same checks (65 tasks, 7 probes, 171 launches).

Scope. Arguments are not decided or recorded: they are in the agent's memory and another thread may change them after they are read. An allowed interpreter runs any code the agent

can read. A swap between allowed programs is killed at the swapped program's first mediated call, not prevented. Shared libraries are not decided: Landlock governs launches, not the executable mappings of libraries, so where a policy lets the agent both write and read a directory, a library it writes may run inside an allowed program; v1.27.1 closes this for files (§2.23).

2.23 Nothing written during the run is mapped executable (v1.27.1)

v1.27.0 keeps the agent from launching a file it wrote, but not from loading one: a shared library written to a directory the agent may also read could be mapped executable into an allowed program (`LD_PRELOAD`, `dlopen`, a native extension). Such a library runs under the same filter, Landlock domain and rules as the agent, so it escapes no containment; what it breaks is the claim that nothing written during the run runs as a file, and the meaning of a launch record. A `noexec` mount in the agent's own namespace would not help: the agent opens no file itself. The Warden opens each file it allows and hands over the descriptor, and the kernel checks an executable mapping against the mount the file was opened through, so the mount has to be where the Warden opens. From v1.27.1, under `require warden 1.27`:

1. **Which directories.** A path rule is writable when an open its flag clause admits writes (write or read-write access, `O_TRUNC` or `O_CREAT`), computed from each rule as both parsers read it, so they agree on the list by construction. Each names a directory: a prefix's (its parent without a trailing `/`), the directory before a glob's first wildcard, or an exact rule's file (its directory if it is not there yet). A rule no directory bounds, or a writable `/`, refuses the start.
2. **The mounts.** Before it starts any helper or the agent, the Warden makes a private mount namespace, so nothing reaches the host, and re-mounts each directory on itself `noexec` with the flags it already has, with every mount beneath it, then confirms each. It re-enters its working directory through the new mounts, and refuses to start if a descriptor the agent would inherit is open for reading on a file there, or if the agent's program, its interpreter or a loader a launch needs is under one.
3. **What the agent sees.** `mmap` with `PROT_EXEC` of a file there fails `EPERM`, `mprotect` adding it `EACCES`, and its launch is refused by the kernel; a copied library cannot be loaded and the process goes on. Anonymous executable memory (a JIT's), pure-Python modules and `ctypes` callbacks are unchanged.
4. **Fail closed.** A descriptor for a path under a listed directory must come through a `noexec` mount (`fstatvfs`) or the open is refused, so a directory the host makes during the run, or a file the host replaces, is refused rather than left executable.

5. **Recorded and audited.** `run_start` records each directory, the policy line that makes it writable and that rule's own directory, or `null` when the operator turned the mounts off for a run (`--writable-exec`, noted by the audit). `varek_audit.py` checks each entry against its rule, every writable rule against the list, and every write the stream records against the list.

Measured. A run's startup grows with the number of writable directories, about 0.2 ms a directory at 50; an open under one is within the spread between runs. `make test-v1271` (73 checks) runs a probe as the agent with copies of host libraries. A 24-hour run on a 1-vCPU, 1 GB host, with the work directory mounted `noexec`, passed: 1,296 tasks (`git`, `python3`, a compile) all succeeded, 144 probes all refused and recorded, among them 36 copies of a library the agent wrote and tried to load, and the audit passed on the 282,917-record stream.

Scope. Code the agent writes into memory and runs there is not covered, and is not claimed. A mount of a writable directory that the host made elsewhere before the run is not under a listed directory. A compiled package installed into a writable directory during the run cannot be loaded without `--writable-exec`.

2.24 The agent holds only placeholders (v1.28.0)

An agent that calls an API needs its key, and through v1.27.1 the key was in the agent's memory, where anything the agent ran could read it and send it anywhere a rule allowed. Deciding the request does not help once the key is in the process: a request rule can say where a request may go, but not keep the agent from putting the key in an allowed request's query, or into a file it may write. v1.28.0 keeps the key out of the process. Under `require warden 1.28`:

1. **The grammar.** `credential NAME from STORE:REF` names a credential (`file:` an absolute path, `vault:PATH#FIELD`, `aws-sm:ARN`); on an `allow request` rule, `credential=NAME header=HEADER[:SCHEME]` says where it goes. The rule must be `https://`, and framing headers are refused. The decision procedure, the certificate checker and the cross-check's oracle each read it; the cross-check's generator writes valid and malformed credential policies, and the three agree.
2. **Fetching.** `warden-secrets`, its own binary, loaded into a sealed memfd and hashed into `run_start` like the proxy, reads its configuration and local stores as root (each a root-owned file readable by no one else, under directories no other user can write), then runs as its own user with no capability, no new privileges and not dumpable, and fetches over verified HTTPS: Vault (KV, or a leased secret it renews) and AWS Secrets Manager (SigV4, from a key file or the instance's role). The Warden learns each value's version

and length, never the value, and refuses to start when the policy would let the agent open any file the helper read, by its own path or by a second name (a hard link, a bind mount).

3. **The swap.** The agent gets a placeholder made for the run in `VAREK_CRED_<NAME>`. The values go from the helper to `warden-proxy` over a socket the Warden passes on without reading; the proxy holds them in locked memory. In inspecting mode (§2.21), on a request whose deciding rule names a credential, the proxy replaces the placeholder in that header, which must occur once and hold exactly the placeholder (after the rule's scheme, if it names one), or refuses the request before a byte is sent. A placeholder anywhere else is sent as it is, so the value reaches only the server and header a rule names.
4. **Revocation.** `varek revoke <run>`, through a control socket only root can reach, makes every later mediated call of the run UNSATISFIED, including connects in progress and the stub resolver's answers; the proxy clears every value and closes every connection; the helper revokes the run's Vault leases. The run goes on, held to refusals, so what the agent tries after it is recorded.
5. **Recorded and audited.** `run_start` records each credential's store, reference, version, length and placeholder's SHA-256; a request that carried one records its name and header. `varek_audit.py` checks each such request against its deciding rule, every request such a rule allowed against the credential it must have carried, each refused swap against its request, and every decision after a revocation against UNSATISFIED.

Measured. `make test-v1280` (168 checks, 155 of which fail against v1.27.1) runs against stand-ins for Vault and AWS Secrets Manager that check each request's signature, and a TLS server that logs what reaches it: the value arrives where a rule names it and nowhere else. A soak trial on a 1-vCPU, 1 GB host, four credentials, revoked near the end, passed.

Scope. A server the policy lets a credential reach can send the value back in a response the agent reads; the policy must not allow such an endpoint with a credential, and the proxy does not scan responses, which could not see through compression or an encoding. What was sent before a revocation stays sent. The proxy holds the values for the run. Only headers, only in inspecting mode.

2.25 Who each decision was for (v1.28.1)

An agent acts for a user, and it may start other agents that act for it. Through v1.28.0 a verdict said what was done and which rule decided it, but an auditor asking for whom had a process id and nothing else, and anything the agent said about its user or its sub-agents

would be the agent's own claim. v1.28.1 records the chain from evidence the agent does not control. Under `require warden 1.28`:

1. **The grammar.** `trust issuer URL audience AUD [jwks URL]` names an identity provider whose tokens the run accepts for the user (at most 8); `trust spiffe DOMAIN bundle PATH` a SPIFFE trust domain and its bundle; `require on-behalf-of` refuses a run with no verified user; `subagent=NAME` on an `allow exec` rule makes the program it launches a sub-agent. The decision procedure, the certificate checker and the cross-check's oracle read them alike; a v1.28.0 Warden refuses them.
2. **The user.** The operator gives the run the user's token, a JWT (possibly from an OAuth 2.0 token exchange), in a root-owned file. `warden-secrets` (§2.24) fetches the issuer's keys over verified HTTPS at each start and checks the signature (RS256, PS256, ES256 or EdDSA; `none`, every HMAC algorithm, `crit` and keys the token carries refused), the issuer, the audience, the times (at least 60 s before the expiry), a canonical encoding, and the `act` claim as the token exchange's actors (at most 8 deep). Only the verified claims reach the Warden; the token reaches no output, stream, export or agent environment.
3. **The agent.** An X.509-SVID and its key, checked against its trust domain's bundle: one SPIFFE ID in the domain, a leaf that is not a CA, the chain and the key. Every bundle the policy trusts, the token and the SVID must be out of the agent's reach for writing, by any name.
4. **Sub-agents.** The Warden makes a cgroup v2 group for the run and, at each launch an `allow exec ... subagent=NAME` rule decides, a group for that sub-agent nested under its launcher's, and moves the launching process into it before the launch is answered. Every call is then attributed by the group the kernel reports for its process (through a pidfd where the kernel gives a cgroup id, from `/proc` otherwise), so a sub-agent's calls, and those of everything it starts, count as its own whatever it says. A launch counts once the kernel's process events say it happened; a failed one is moved back to its launcher. A threaded launcher, a call from outside the run's groups, and a group that cannot be read are refused. The agent runs unprivileged and cannot write a control-group file.
5. **Expiry, records and audit.** When the user's token expires during the run the Warden records it and revokes the run as `varek revoke` does (or, by the operator's choice, records it only). `run_start` records the verified chain and every decision record its `obo` (`["user", "agent", "subagent:coder#1"]`). `varek_audit.py` checks the chain against the policy's issuers and trust domains, every record's `obo` against the launch that started its sub-agent, the records' time order, and the expiry and its revocation.

Decisions do not depend on the chain: v1.28.1 records and audits it, and a rule on it (`for=subagent:NAME`) is left for a later release. Recording cannot move any action to

SATISFIED.

Measured. `make test-v1281` (208 checks, 197 of which fail against v1.28.0) runs against a stand-in identity provider and SPIFFE authority whose tokens and SVIDs are signed by an implementation apart from the C verifier, with each check of a token or SVID failing on its own. Attributing a call by its group adds −1 to +4 µs to the p50 decision time through a pidfd and 4 to 9 µs through `/proc`; a run with a user's token starts in 38 ms against 16 ms without (keys fetched from a local stand-in).

Scope. The token is checked at the start: a token its issuer revokes during the run is not noticed, its expiry is. Its `typ` is not checked. A compromised issuer or trust-domain CA can name any user or agent. Records that are not a process's call carry no `obo`. An X.509-SVID from a file only. The operator is recorded as the host reports it, not verified.

2.26 Running an agent with nothing set up first (v1.29.0)

Every guarantee above starts from a policy, and through v1.28.1 someone had to write one before the agent could run. Most people do not know what their agent touches. v1.29.0 writes the first policy for them, shows them what a narrower one would refuse without enforcing it, and drafts the next one from the record. Under `require warden 1.29`:

1. **Opening as the agent.** Through v1.28.1 the Warden opened every allowed file as root and handed the agent the descriptor, so the policy was the only check. Now the walk and the open (and every change by name, below) are made by a Warden thread holding the agent's uid, gid and groups, so a call needs both the policy and the kernel's permission checks for the agent's user, and the agent's files are its own. The decision is still made, and certified, on the canonical name before anything is done.
2. **Changes by name.** unlink, rmdir, mkdir, rename, chmod, utimes, truncate and chdir are each decided as a write on every name they touch (a rename on both) and carried out as the agent; link and symlink stay refused, since a second name is how a deny rule would be escaped. A mkdir of a name that is there is answered EEXIST where the agent may look the name up, and a lookup under a missing directory is decided on the whole name asked for.
3. **The baseline and the profiles.** With no settings, `varek run` writes the policy for the run from the host as it is: the credential stores and keys refused first, the system read-only, the working directory and a private temporary directory writable, the agent user's home closed but for its profile's directories, launches of the programs root owns by name, no network but the profile's hosts. It is an ordinary policy, stored by its SHA-256 and recorded in `run_start`. The agent runs as the user who ran sudo, never root. The profiles

(Claude Code, the Codex CLI, the Gemini CLI, Aider, Python, Node.js) were built from dry runs of the real agents and checked by running each enforcing against the same command without VAREK. Doing so found what no test had: inside a directory mounted `noexec` (§2.23) the Warden now keeps exec where no rule that lets the agent write can reach (every path a rule matches starts with its constant before the first wildcard), so an agent installed in a home it may write in can start; stat(2) and lstat(2) are decided (musl calls them); and a short list of calls on the caller itself is admitted.

4. **The dry run.** With `--dry-run`, every call is decided and recorded and one the policy refuses is let through: the kernel carries it out as the agent (seccomp user-notify's CONTINUE), or the Warden dials a refused connect. The hard-denied system calls, the protected objects, Unix-socket connects and revocation still refuse, and a dry run is refused with a plan, credentials or sub-agents. Each call let through is recorded with the verdict the policy gave and `"enforced":false`; `run_start` and `run_end` say `dry_run`, and `run_end` counts them.
5. **Drafting a policy.** `varek policy suggest` reads a run's stream and writes a draft: exact rules for what was refused, each re-decided by the decision procedure, and commented out, never active, anything a deny rule refuses, a credential store or key, another user's home, a write under a system directory, a launch of sudo, su, doas or pkexec, and any program the agent wrote. A person reviews it.
6. **Audit.** `varek_audit.py` holds a dry run's records to what a dry run may let through and verifies such a stream with exit 3, never 0, so a dry run cannot be read as enforcement; it checks whom files were opened as, the baseline's header against the policy file, every change by name, and every place kept executable against the rules. The CycloneDX export marks the mode on every authorization it lists.

None of this changes a decision: the baseline, the profiles and the drafts are policies decided by the same procedure and certified by the same checker, and a dry run's records keep the verdict the policy gave. No extension may move a genuinely unsafe action to SATISFIED.

Measured. `make test-v1290` (316 checks; 290 of the 313 of the tests-and-regression step fail against v1.28.1). Opening as the agent adds 11 to 19 µs to the p50 decision time of an allowed open and 5 to 7 µs to a refused one; the baseline and profiles (205 to 221 rules) cost no measurable time against a 34-rule pack, since the procedure compiles the rules before the run; in a dry run a system call outside the allowlist takes 51 µs against 0.4 µs refused. The launch set is hashed with OpenSSL's libcrypto, which reads no configuration: a run under the baseline starts in about 170 ms, and with a Node.js agent's profile in 540 to 650 ms on a CPU without SHA extensions (230 and 920 ms with libsodium's SHA-256).

Scope. A dry run is not a sandbox: in one, the agent can do what its own user can, outside the fixed refusals. A program root does not own (Node.js under nvm) is not named by a profile. inotify, statfs(2) on a path, link, symlink and listening sockets stay refused under the baseline. The adversarial review by AI review agents has not been done for this edition (§5).

2.27 MCP servers, MCP calls and MCP tokens; OCSF events (v1.31.0)

An MCP server is a program the agent starts and talks to (over stdio), or a service it calls over HTTP. Through v1.29.0 a local server was decided by the agent's own rules, so a filesystem server could read whatever the agent could; a call to a remote one was a `POST` to one URL, so a policy could not allow one tool and refuse another on the same server; and an MCP token was a credential like any other. Under `require warden 1.31`:

1. **Confined subjects.** `mcp server NAME exec PATH [sha256=HEX]` declares a server (`subagent=NAME confined` does the same for a v1.28.1 sub-agent). When the agent launches it, it runs in a control group of its own (§2.25), and every call it makes, and every call of every process it starts, is decided by the SMT decision procedure against the rules written `for=NAME` and no others: the agent's rules do not apply to it, and its rules do not apply to the agent. The subject is an input to the decision and is named in the certificate, and the checker replays a certificate under that subject only. The hard-denied calls, the protected objects and revocation hold for a server as for the agent. The start is refused when a server's file, or a directory above it, is one any subject's rules let be written, when its SHA-256 is not its pin, or when it is, or reaches, an interpreter or a launcher (`node`, `npx`, `docker`, a symlink to `python3`). A confined launch whose environment sets a variable that makes a loader or interpreter take code from a path or a value (`LD_PRELOAD`, `PYTHONPATH`, `NODE_OPTIONS` with `--require`, `BASH_ENV`, ...) is refused, checked in the launcher's memory at the decision and in the new image's own copy before any of its calls is answered: the launcher would otherwise choose code that runs under the server's rules.
2. **MCP calls over HTTP.** Under inspecting mode (§2.21), `allow|deny mcp URL METHOD [NAME]` makes URL an MCP endpoint. The proxy holds each `POST` to it, reads the whole body (up to the rule's `max_body`), parses it with a strict JSON parser as one JSON-RPC message (no duplicate members, UTF-8 only, no batch), refuses one sent other than as `application/json` in UTF-8 or whose `Mcp-Method` or `Mcp-Name` header disagrees with it, and the Warden decides the object `<canonical URL>#METHOD#NAME`. Nothing of the request reaches the server before the call is decided and certified. Another spelling of an endpoint (its case, a trailing `/`) is refused, not decided by request rules. Each call is recorded with its method, name, id and the body's length and SHA-256.

3. **MCP tokens.** A credential with `audience=URL` (§2.24) is an MCP access token: at fetch, `warden-secrets` checks that it is a JWT whose audience is that server's canonical URL and that it has not expired, and it is released only into calls to that server. On every MCP call the bearer token the agent sends itself is checked: refused if it is the run's user token (by its bytes, or by its signing input, since an ECDSA signature can be re-made without the key), if it is a JWT for another audience or expired, or if it is in a JWT's form but not a strict compact JWS (a lenient reader would take a padded one for the same token). `trust mcp-resource URL` lets the run's user be named by a token whose issuer is found through the server's protected resource metadata (RFC 9728). Tokens come from stores; VAREK runs no OAuth flow.
4. **OCSF events.** `varek export --format ocsf` writes each record of an audited stream as an OCSF 1.9.0 event (file, process, network, HTTP, DNS and API activity), with the decision in the `security_control` profile, the agent, its user and its sub-agents in the `ai_operation` profile, and each event's record report id, sequence number and chain hash, so `--verify` can check events against the stream (and audits the stream to do so). A dry run's would-be refusals are Denied and Detected, never Blocked. `--follow` writes events as the Warden writes records.
5. **Audit.** `varek_audit.py` checks that each decision cites a rule of the subject it was decided as, and its certificate's rule; that an MCP token is released only on a call to its own server; that no token value appears in any record outside what echoes the agent's own requests; and every `net.mcp` record's endpoint and certificate.

None of this changes a decision outside what the new lines decide: confinement and the token checks only refuse, and an OCSF event reports a decision, it decides nothing. No extension may move a genuinely unsafe action to SATISFIED.

Measured. `make test-v1310` (317 checks; 263 of 271 fail against v1.29.0, the 8 that pass holding trivially there). A confined server's decision costs no more than the agent's (103 µs against 126 µs p50 for an allowed open after the baseline); an MCP rule adds nothing to a small call and about 9.6 ms per MiB to a large one, since the body is held whole before it is sent; the decision itself stays under 30 µs. A run starts in about 170 ms under the baseline, 250 ms with one MCP server and 420 ms with four. The rule cap is 512 (256 before); a decision's work stays bounded by the 4,096 glob tokens of §2. A 24-hour run on a 2 GB host passed: two confined MCP servers kept for the run, 1,291 tasks before the user's token expired, all succeeding, 144 probes held, the MCP token reaching the remote server in exactly the 258 calls a `for=gh` rule allowed, the run revoked at the expiry, and 11,719 OCSF events followed live and verified against the 13,891-record stream.

Scope. The agent's calls to a local MCP server are bytes on a pipe and are not decided; a confined server bounds what any call to it can do. A call's arguments and a server's responses are not decided. A server that reads code or settings from a file another subject may write runs what was written there, under its own rules; the environment check is a list of variables that name code, not an allowlist. The review by six AI review agents found three high findings, all fixed before this edition (§5).

2.28 Containers: an official image, Docker and Podman (v1.32.0)

Most production agents run in containers, where through v1.31.1 the Warden could not start under the runtime's defaults (it needs CAP_SYS_ADMIN for the agent's namespaces) and the usual answer, `--privileged`, grants everything and turns off the container's own seccomp profile and its AppArmor or SELinux confinement. From v1.32.0:

1. **An official image.** `ghcr.io/kwdoug63/varek`, built on Red Hat's UBI 9 minimal with no compiler, carries the Warden, its helpers, the tools and the shipped policies. Its entrypoint is `varek run --`: the container's first process is the Warden, and the agent is its child in PID, network and mount namespaces the Warden makes inside the container, so every decision, record and refusal is the one it makes on a host.
2. **A published, minimal set of settings.** `--cap-drop ALL` and ten capabilities added back (SYS_ADMIN, SYS_PTRACE, SETPCAP, SETUID, SETGID, DAC_OVERRIDE, KILL, NET_ADMIN, NET_BIND_SERVICE, CHOWN), each with the reason the Warden needs it and each shown needed by a suite that fails without it; `--cgroupns=private`; the runtime's default seccomp profile, unchanged. The agent holds none of them: its bounding set is emptied and it runs as its own user.
3. **Confinement kept on.** Docker's `docker-default` AppArmor profile denies every mount, and the Warden mounts the agent's writable directories `noexec` (§2.23). VAREK's profile is `docker-default` with the Warden's three kinds of mount allowed (its own namespace made private, a directory bound, a bind mount's flags changed) and no filesystem mounted or reconfigured. On SELinux the container runs as `container_init_t` with a module of four rules (the cgroup filesystem's remount; mounting on the container's and the host's `/dev/null` and `/dev/tty`).
4. **Sub-agents and confined MCP servers in a container.** The runtime mounts the cgroup filesystem read-only. With a cgroup namespace of its own, the Warden bind-remounts that view read-write in its own mount namespace, changing only that mount's flags, never the filesystem's, which every cgroup2 mount on the machine shares; outside its namespace's root it refuses. In a container the kernel's process events do not reach the Warden, and through v1.31 a sub-agent of the image its launcher runs was then refused,

since its outcome could not be told. From v1.32.0 such a launch is told by the process's memory layout (where its code, stack, heap, arguments and environment begin and end), which every new program gets anew, randomized, and which a process cannot change except by a launch (the agent's filter refuses `prctl(PR_SET_MM)`, under every policy); the same layout is a failed launch, and where layouts are not randomized the launch is still refused. A Python agent can launch a Python MCP server confined.

5. **Checked, and refused rather than run with less.** `varek doctor` in a container checks each setting and makes the Warden's mounts in a child, naming the flag that supplies anything missing; `varek run` refuses a container that cannot carry the run (a missing capability always; the profile, the mount test and the cgroup namespace when the run needs them). Rootless Podman needs `--systemd=always` for sub-agents, and is refused without it.
6. **A signed image with its bill of materials.** Each release tag on main builds the image, checks it (`varek doctor`, an agent's run, Docker's defaults refused), pushes it, signs it with cosign keyless (the release workflow's own identity, in Sigstore's public log) and attaches a CycloneDX SBOM as a signed attestation; both are verified before the `:1.32` and `:latest` tags move to it.

None of this changes a decision procedure, a rule or a record: the Warden in a container decides as on a host, and the one launch decided differently (a sub-agent of its launcher's image, without process events) is now made where it can be told and refused where it cannot. No extension may move a genuinely unsafe action to SATISFIED.

Measured. Every suite passes inside the container, 31 of 31, on Docker 29.1.3 on Ubuntu 24.04 (kernel 6.8, AppArmor) and on rootful and rootless Podman 5.8.2 on RHEL 9.8 (kernel 5.14, SELinux enforcing). `make test-v1320` adds 25 checks (the version, `varek doctor` and the run's refusal in a container, the packaging and the release workflow's pins and gates). The shared 24-hour soak passed inside a container under the documented settings and nothing more: 1,440 turns in each run, every task done, every probe held, the user's token expiry revoking the run 0.02 s after, and memory flat.

Scope. The container holds CAP_SYS_ADMIN, the agent never does; running the Warden outside the agent's container is v1.34's node agent. The AppArmor profile bounds what is mounted, not where: a process with CAP_SYS_ADMIN in the container may bind any path and make a bind mount read-write. Container detection knows Docker and Podman. The review by six AI review agents found five high findings (the release workflow's check that could not fail and its actions on movable tags, an image build that could take binaries built on the host, `varek run` refusing every run under `docker-default`, and a test's skip decided by the Warden

under test) and a cgroup remount that reconfigured the filesystem every cgroup2 mount shares, all fixed before this edition (§5).

2.29 Kubernetes: a node agent and a Helm chart (v1.34.0)

v1.32 runs the Warden inside the agent's container, which needs ten capabilities, VAREK's confinement profile and a private cgroup namespace in the pod; a cluster's admission policy (Pod Security's "restricted" and "baseline" levels, OpenShift's default constraints) refuses that pod. From v1.34.0 the Warden can supervise a pod's containers from outside them, and the agent's pod passes "restricted":

1. **A node agent.** One Warden on each node (a DaemonSet), privileged on the node as node-level security agents are, listening on a socket the container runtime dials. The container's seccomp filter is the Warden's own for its policy, in the OCI form, with `SCMP_ACT_NOTIFY` and `linux.seccomp.listenerPath`: the runtime hands the filter's notification descriptor and the container's state to the node agent as it creates the container, and from then on the container's calls are decided as the Warden decides an agent's on a host. Each container has its own supervisor, policy and verdict stream. The node agent takes a connection only from root running where the runtime runs, and a descriptor only for a process the runtime started.
2. **An NRI plugin** (`varek-nri`; containerd 2.2 or later, CRI-O 1.35 or later). For each container of a pod annotated `varek.dev/policy: NAME`, it sets the Warden's filter (written by the Warden from the policy, which the node agent loads again and holds to its SHA-256), puts the launcher first in the command, and makes every writable mount `noexec`. A container it cannot carry is refused, with the reason in the pod's events.
3. **The launch set in a pod.** Landlock cannot be added to a running process from outside, so a small static launcher, `varek-init`, bound read-only at `/varek/init`, applies the launch set (§2.22) and runs the container's command; the node agent checks it is that file. A policy that decides launches needs a read-only root filesystem.
4. **A container that is not adjusted does not run.** Each container of a supervised pod names VAREK's unsupervised seccomp profile, which refuses every call and which only the plugin replaces. With no plugin on the node, an older runtime, or a plugin that does not answer in time, the container keeps that profile and does not start.
5. **The pod's network.** The agent has the pod's network namespace; every connect and send is decided as on a host, and the stub resolver (§2.19) and the egress proxy (§2.20) run inside the pod's namespace, entered by the node agent.

6. **Policies as cluster resources.** `VarekPolicy` (`varek.dev/v1alpha1`), namespaced, holds the policy as written for the Warden. An admission webhook refuses a policy the Warden does not load, with the Warden's reason, and most of what the node agent would refuse at a container's start; a namespace labelled `varek.dev/policy-required=true` refuses pods that name none.
7. **Evidence on the node.** One verdict stream per container, outside every pod, signed with the node's key and anchored, with the policy copied beside it; an exporter gives each as OCSF (§2.27) for the cluster's collector.
8. **A Helm chart**, signed and published with the node image (`oci://ghcr.io/kwdoug63/charts/varek`), pinning the image by the digest the release signed, with OpenShift's security context constraint for the node agent.

None of this changes a policy line or a decision procedure: a container under the node agent is decided as an agent on a host. The policy grammar stays at 1.31 (`require warden 1.34` is refused). No extension may move a genuinely unsafe action to SATISFIED.

Measured. A 24-hour run passed on Amazon EKS (two m6i.large nodes, Amazon Linux 2023, Kubernetes 1.35.8, containerd 2.3.5): 2,592 turns of work and 576 git commits by the long agents, 97 short Jobs, 3,465 requests to the allowed service (p50 3 ms, p99 18 ms at the agent) and none to the denied one, every probe held (`/etc/shadow`, an unnamed volume, a program the agent wrote, an unnamed service, the instance metadata), and 99 streams of 1,246,748 records all audited. A 24-hour run also passed on single-node OpenShift 4.22.18 (CRI-O 1.35.9, SELinux enforcing): 1,296 turns, 2,160 allowed requests and none denied, every probe held, 97 streams of 807,885 records all audited.

Scope. The admission webhook is part of the guarantee: it requires each supervised container to name the unsupervised profile, and an annotated pod in a namespace it does not check (`kube-system` by default) is not checked. The pod's other containers share its network namespace. The node's signing key is one Secret given to every node. Credentials (§2.24), sub-agents and confined MCP servers, `require on-behalf-of`, plans and the dry run are not yet carried under the node agent; a policy that needs them is refused at deploy and at the container's start. The node image is amd64 only. The review by seven AI review agents found, among others, a supervised pod that could run unsupervised where the plugin did not adjust it, and a node-agent connection taken on a parent process's word; both were fixed before release, and every finding is listed in `RELEASE-v1.34.0.md`.

3 Three-state decision semantics

The three-state result is the core of VAREK's safety claim and the reason the system is honest about its own limits.

RESULT	MEANING	RUNTIME BEHAVIOR
SATISFIED	Provably compliant	Proceed
UNSATISFIED	Provably non-compliant	Deny
UNKNOWN	Not provable within bounds	Fail closed (deny)

A two-state system is forced to convert every UNKNOWN into either a false pass or a false block. VAREK refuses that conversion: it reports UNKNOWN as UNKNOWN and lets the fail-closed runtime resolve it safely. In the running agent an UNKNOWN is refused, like an UNSATISFIED (symmetric suppression). At the plan gate with a flow policy, it goes to the deterministic disposition that policy declares, and the progress-safety proof certifies that every such refusal ends in an automated outcome.

3.1 Soundness, and why UNKNOWN is the honest residue

VAREK is **sound but deliberately incomplete**. Soundness: no action is reported SATISFIED unless it provably satisfies policy. Incompleteness: some safe actions cannot be proved safe within the decision procedure's bounds and are reported UNKNOWN. The asymmetry is intentional — over-refusing a safe action is a utility cost; wrongly authorizing an unsafe one is a safety failure. The system is built to never make the second trade.

This is also why the input space being effectively infinite is not a problem the way enumerating edge cases would be. The decision procedure reasons over whole domains symbolically rather than sampling points, and the three-state verdict is *total*: every input lands in exactly one of SATISFIED / UNSATISFIED / UNKNOWN, deterministically, with UNKNOWN as the fail-safe residue. Coverage of the infinite space is by construction, not by enumeration.

3.2 UNKNOWN diagnostics and resource bounds

An UNKNOWN record's `rule` field says why: `fragment_escape_flags`, `fragment_escape_length`, or `default_deny_unknown` when no rule matched (from v1.13.0). The procedure's work is bounded by construction — a length guard on the object (4,095 bytes), a bounded

enumeration of symbolic flag bits (16), and from v1.16.0 a cap of 4,096 glob tokens per policy, which bounds the steps of one decision. There is no wall-clock ceiling and no memoization cache; a bound is structural, so a decision that would exceed it is UNKNOWN before it starts rather than cut off mid-way.

3.3 Measuring the residue

The verdict-distribution harness (v1.13.0) measures what fraction of safe actions a policy clears and gates every change on `unsafe_satisfied == 0`. On its synthetic seed corpus of file opens, v1.13 cleared 85.4% of safe opens with no unsafe open authorized, where the example policies as v1.12.4 shipped them cleared 75.6% and authorized 24 unsafe opens. These are figures on a synthetic seed corpus, not on a customer workload; a baseline on a customer workload is future work.

---

4 Verification scope and guarantees

VAREK verifies that agent actions and plans satisfy explicitly authored policy, that data flow across declared plan steps respects explicitly authored information-flow constraints, and that an unattended policy is progress-safe. The guarantees are relative to the policy as written and to the fidelity of the action model derived at the system boundary.

What the Warden decides. File opens, file lookups, outbound connects, datagram sends and the agent's launch are held and decided. Agent-initiated program launches after the first are refused whatever the policy says unless it opts in with `require warden 1.27` (§2.22), and sends that name a destination are refused (except a question to the v1.25 stub resolver) and inbound networking. System calls the filter admits without asking (memory, time, threads, and reads and writes on descriptors the agent already holds) run undecided, and every other system call is refused outright.

Independence from the agent platform. The decision procedure and the Warden run outside the supervised agent and do not depend on its model provider, agent framework, or orchestration platform: the action model is derived at the system boundary, and enforcement uses seccomp-BPF and seccomp user-notify, which apply to any supervised process. The verdict stream and its CycloneDX 1.6 export are produced by the Warden, not by the agent or its vendor, and can be signed and anchored, so a third party can inspect and verify the authorization record without relying on the vendor that built the agent. The dependency is on the host kernel (Linux with seccomp user-notify) and on the Warden's

trusted computing base, stated in `docs/security/TRUSTED-COMPUTING-BASE.md`; it is not on any particular model or agent platform.

Declarations are not observations. Plan edges, plan-step fields (v1.20) and declared open flags (v1.21.1) are the agent's declarations. The plan gate decides what the agent declares; the runtime decides what the agent does. Nothing compares the agent's later calls with its declarations, and every call is decided again at run time.

Explicit information flow. The data-flow subsystem covers explicit flows — data that moves through observable action inputs and outputs. Implicit flows (information conveyed through control structure rather than data movement) are on the roadmap and are not claimed. An allowed network destination is a channel; what the agent sends over it is outside the Warden's decision, except that in inspecting mode (§2.21) each request's method, path and query are decided and its body bounded by length. The published `docs/security/threat-model.md`, `docs/security/threat-model-dataflow.md`, `docs/security/bypass-classes.md`, and `docs/security/TRUSTED-COMPUTING-BASE.md` are the authoritative statement of assumptions, adversary models, in-scope and out-of-scope threats, the per-class mediation status, and the per-component trusted-versus-verified status of the verification chain. Every release's notes list its known limits.

---

5 Quality and testing

The testing posture mirrors the runtime posture: where a guarantee cannot be established, the build fails closed rather than presenting an unverified result as a passing one. Each release adds a regression suite that fails against the previous Warden, and every earlier suite must still pass.

- Progress verifier: `test_v19_progress.c`, 10/10, clean under `-fsanitize=address,undefined`.
- Enforcement, measured directly: the TOCTOU race harness reports 1,848 to 1,889 leaks in 20,000 attempts for approve-then-continue versus 0 for resolve-and-inject (v1.18.0 re-run). io_uring refusal under the live filter is asserted by `test_v14_filter`.
- Mediation completeness on the live kernel: `test_v192_abi_lockdown.c` admits the native call and kills the x32 call (release-blocking); `test_v192_baseline_deny.c` confirms the hard-deny set and `CLONE_NEWUSER` denial. `target_conformance` passes every phase under the live Warden.
- Lifecycle coupling: `test_v193_lifecycle.c` kills a live supervisor and asserts no agent process survives.

- Mediation correctness: `make test-v112` through `make test-v1124` — an adversarial target exercising traversal, symlink, `/proc/self` and record-injection escapes; side-effect-free denial; threads and children; dynamically linked agents; the plan gate.
- Decision procedure: `make crosscheck` (zero disagreements with the off-the-shelf SMT solver over 26,624 checks at v1.13), the derivative-based reachability oracle (v1.14), and the verdict-distribution harness's `unsafe_satisfied == 0` gate. `make test-v1130`, `test-v1140`, `test-v1150` (including a planted procedure bug the checker must stop).
- Record integrity and deployment: `test-v1160` through `test-v1162`, `tools/varek_preflight.sh`, and a CI job that builds the Warden with libseccomp and libsodium, lints the shipped policies and runs the preflight on each.
- v1.17.0: 48 checks (protected objects by identity, mediated lookups, unprivileged agent). v1.18.0: 73 checks plus the v1.7 layer's test. v1.19.0 and v1.20.0: breaker session limit and plan-step fields.
- v1.21.0: `make test-v1210`, 90 checks, running curl, Python `requests` (including verified TLS), raw Python TLS, Python Unix and UDP sockets, and Node.js `http`, `https`, `tls`, `net` and `dgram` as the agent under the Warden, plus the 2,000-attempt destination-swap race.
- v1.21.1: `make test-v1211`, 33 checks, 23 of which fail against the v1.21.0 Warden.
- v1.22.0: `make test-cli`, the `varek` command and `varek bench`, including end-to-end runs under the Warden and a bench whose every verdict is checked.
- v1.23.0: the AMI build proves the installed image before it is published: a real run under the Warden, its audit and a signed export, then the bench's smoke run.
- v1.23.1: the license check run against the real AWS License Manager in a second AWS account, through a private offer; each check licensed and returned its unit. Then end to end as a buyer: the published v1.23.1 AMI launched from the listing in that account on an m7i-flex.large with the documented instance role, where `varek license` reported `licensed: enterprise_tier_a`, `varek init --pack hipaa` installed the HIPAA pack, a second `varek license` was licensed again (the unit had been returned), and `varek doctor` reported the host ready.
- v1.24.0: `make test-v1240`, 108 checks, 100 of which fail against the v1.23.1 Warden, parsers and audit: the name grammar in both parsers; the resolution table against a local authoritative test server that rotates answers, follows CNAME chains, truncates, fails and drops queries; Python, curl, Node.js, a static musl binary, Go and Java as the agent, each resolving through the views; rule order, grace, and forged streams the audit

must refuse. `make crosscheck` with name rules, 0 disagreements. A 24-hour soak against CDN-hosted APIs (§2.18) and a latency comparison of numeric and name rules.

- v1.25.0: `make test-v1250`, 161 checks, 129 of which fail against the v1.24.0 Warden, parsers and audit: the wildcard grammar, budgets and acknowledgment in both parsers; the shared-domain rule against an independent statement of it on 17,840 suffixes; Python, curl, Node.js (`dns.lookup` and `dns.resolve4`), Go, Java and a static musl binary resolving through the stub, with a recording upstream server confirming that names outside every rule send nothing; a DNS-tunnel style client held to the budgets; 40 names on one address; and forged streams the audit must refuse. `make crosscheck` with wildcards, budgets and the acknowledgment, 0 disagreements. A 24-hour soak against Wikipedia (§2.19) and latency figures.
- v1.26.0: `make test-v1260`, 156 checks, 147 of which fail against the v1.25.0 Warden, parsers and audit, and 7 of which are not run there (they test parsers v1.25.0 does not have): the proxy grammar in both parsers; the proxy process's user, capabilities and descriptors; synthetic addresses; the hand-off; real ClientHellos from Python, `openssl s_client`, curl and Node.js; decisions against real TLS and HTTP servers, pipelined requests and IPv4-mapped answers; close records with the proxy killed mid-relay; chaining to Squid; and forged streams the audit must refuse. `make fuzz-proxy-parse`, 65 vector checks and 4,000,000 mutated inputs under ASan and UBSan with no fault. `make crosscheck` with the proxy directives, 0 disagreements. A 24-hour soak against three content networks (§2.20) and latency figures.
- v1.26.1: `make test-v1261`, 193 checks, 188 of which fail against the v1.26.0 Warden, parsers and audit (and 3 test `warden-proxy` itself, which v1.26.0 does not have): the request grammar in both parsers, with decisions and certificates checked; `warden-proxy` loaded from a sealed copy; the run's CA, its name constraints and the trust views; terminating TLS with Python, curl, Node.js and Java as the agent, a Squid upstream, servers of the wrong name, self-signed or presenting a CN alone; request decisions, pipelining, queries, fronting and `max_body` against servers that hash what reaches them; and forged streams the audit must refuse. The suite also passes with `warden-proxy` under ASan and UBSan. `make fuzz-proxy-parse`, 66 vector checks and 4,000,000 mutated inputs with no fault. `make crosscheck` with request rules, 0 disagreements. `varek bench --proxy`, latency figures and a 24-hour soak in inspecting mode (§2.21).
- v1.27.0: `make test-v1270`, 164 checks: the opt-in in the three parsers; the launch set for each rule form, with denies, symlinks, loaders and the refused forms; launches decided, refused and not in the set; the identity check, by a test hook and by two threads launching at once; 22 forged streams; a compile; and a measured table of what the

kernel's Landlock refuses (`tests/landlock_table.sh`). Of its 134 checks before the review, 123 fail against v1.26.1.

- v1.27.1: `make test-v1271`, 73 checks, 60 of which fail against v1.27.0: the writable directories for each rule form and the refused forms; the agent's own view of its mounts, including a mount beneath one, kept flags and the host's mounts unchanged; a probe run as the agent with copies of `libm.so.6` and Python's `_json` extension, `mmap`, `mprotect`, anonymous memory and `ctypes`, from inside the writable directory too; the host making and replacing writable paths during the run; and 13 forged streams. A soak trial and a 24-hour soak with a library probe.
- v1.28.0: `make test-v1280`, 168 checks, 155 of which fail against v1.27.1: the grammar in two parsers; fetching from stand-ins for Vault (KV 2, KV 1, a lease, a chunked answer) and AWS Secrets Manager (each signature checked) and IMDSv2, and 18 refusals; the swap against a TLS server that logs what it receives, five refused shapes and a placeholder sent as it is elsewhere; a run revoked in mid-run, lease renewal, and root only; 15 forged streams; the tools; and one check for each fix of the review. A soak with four credentials revoked near the end.
- v1.28.1: `make test-v1281`, 208 checks, 197 of which fail against v1.28.0: the grammar in two parsers; verifying tokens of each algorithm and SVIDs against a stand-in identity provider and SPIFFE authority, each check failing on its own (another key, `none`, an HMAC token signed with the public key, the issuer, the audience, the times, `crit`, an `act` chain 9 deep, a non-canonical encoding, an SVID expired, of another domain or authority, a CA, two URIs, a second key), token exchange, and the files out of the agent's reach; sub-agents that fork, double-fork, use threads, nest, launch twice and fail to launch; a token expiring mid-run, revoked and recorded; 17 forged streams; the tools; and one check for each fix of the review.
- v1.29.0: `make test-v1290`, 316 checks (290 of 313 fail against v1.28.1): opening files as the agent; the baseline, each profile and `varek run`, with Python and Node.js agents; changes by name; the dry run, each kind of call let through and each kind still refused; drafts from `varek policy suggest` and runs under them; 18 forged streams; and what running the real agents' CLIs found. `make crosscheck-profiles`, the baseline with each profile, 0 disagreements. `tests/profiles_v1290.sh`, the four CLIs enforcing against unsupervised.
- v1.31.0: `make test-v1310`, 317 checks (263 of 271 fail against v1.29.0): the grammar in the three parsers; confined servers and their children decided by their own rules, the start's refusals and the launch environment; MCP calls over HTTP, allowed and refused, with nothing refused reaching a stand-in server; MCP tokens against a stand-in

authorization server, each spelling of an audience and each lenient form of a token; the OCSF export validated against the vendored schema, `--verify` and `--follow`; 13 forged streams. `make crosscheck` with policies of the new grammar, 0 disagreements.

- v1.32.0: `make test-v1320`, 22 checks: the version and the grammar; `varek doctor` in a container, its rules from a container's state and the Warden's mounts made in a child; `varek run` refused before the agent starts without a capability, naming its flag; the preflight's section; the AppArmor profile's and the SELinux module's rules compared exactly; the image's stages; the release workflow's pins and gates. Every suite inside the container on Docker with AppArmor and on rootful and rootless Podman with SELinux (`tests/container_sweep.sh`), and each of the ten capabilities shown needed by a suite that fails without it.
- v1.34.0: `make test-v1340` (as root, under runc): the node agent, the plugin's adjustments and the launcher end to end with a stand-in runtime, in eight sections, 0 skipped. The chart on a one-node `kind` cluster in CI (the webhook's refusals from a real API server, a pod in a "restricted" namespace supervised, its stream audited and exported); the same pods under CRI-O 1.37.2 (`tests/v1340_crio.sh`); the CRD, the webhook and the policy sync against Kubernetes 1.37.0's API server (`tests/v1340_apiserver.sh`, 25 checks); every suite run under the Warden and under the node agent (`tests/v1340_sweep.sh`); the chart's 19 tests and the Go tests; and the 24-hour runs on Amazon EKS and OpenShift (§2.29).
- From v1.18.0 each release's own code was reviewed independently before tagging, and the findings fixed in the release are listed in its notes. The v1.24.0, v1.25.0, v1.26.0 and v1.26.1 reviews were each done by four AI review agents, the v1.27.0 review by five (three of them stopped partway by a safety classifier), the v1.27.1 and v1.28.0 reviews by three each, and the v1.28.1 review by five, separate from the session that wrote the code, each required to reproduce every finding; they are not a human or third-party review. v1.29.0 has had a correctness review (the code read, GCC's static analyzer, the suite under ASan and UBSan) by the session that wrote it; its adversarial review by AI review agents is pending. The v1.31.0 review was done by six AI review agents, one per area (confined subjects, MCP calls over HTTP, MCP tokens, the grammar in the three parsers, the audit and the OCSF export, and the Agent Baseline mapping against the code); every finding was reproduced and fixed with checks, and is listed in `RELEASE-v1.31.0.md`. The v1.32.0 review was done by six AI review agents, one per area (the Warden's changes, the image and its confinement, the CI and release workflows, the container checks in the tools, the tests and the container sweep, and the documentation against the code); every finding was checked against the code before it was fixed, and is listed in `RELEASE-v1.32.0.md`. The v1.34.0 review was done by seven AI review agents,

each on one part (the node agent, in two parts; the strict JSON reader, the OCI filter and the launcher; the Go plugin, policy sync and webhook; the chart, images and workflows; the tools and tests; the documentation against the code); each finding was read in the code before it was fixed, and is listed in `RELEASE-v1.34.0.md`. A human or third-party review has not been done.

---

6 Demo

A narrated demo walks through the stack end to end: authorization on a compliant action, denial on a violating action, fail-closed behavior on UNKNOWN, and cross-action data-flow scenarios where a sequence is blocked on the basis of where data originated (youtu.be/6E1Yrt02GNM). A browser visualization of the three-state verdict is published on the project site; the runnable C demo is in the repository (`v1_7`, `make demo`; `demo_hootl.c` for the HOOTL walkthrough), and the conformance target runs a real workload under the live Warden. The VAREK Verdict Service at `api.varek-lang.org` answers plan-verification requests with the v1.6 compositional evaluator; its front end, `v1_6/plan_verify_cli.c`, is in the repository from v1.16.3 and remains a demonstration with a built-in policy.

---

7 Documentation

- `README.md` — what VAREK is, the verdict model, and a summary of every release.
- `RELEASE-v1.34.0.md`, `RELEASE-v1.32.1.md`, `RELEASE-v1.32.0.md`, `RELEASE-v1.31.1.md`, `RELEASE-v1.31.0.md`, `RELEASE-v1.29.0.md`, `RELEASE-v1.28.1.md`, `RELEASE-v1.28.0.md`, `RELEASE-v1.27.1.md`, `RELEASE-v1.27.0.md`, `RELEASE-v1.26.1.md`, `RELEASE-v1.26.0.md`, `RELEASE-v1.25.0.md`, `RELEASE-v1.24.0.md`, `RELEASE-v1.23.1.md`, `RELEASE-v1.23.0.md`, `RELEASE-v1.22.0.md`, `RELEASE-v1.21.1.md`, `RELEASE-v1.21.0.md`, and the release notes for each earlier version — what changed, how it was tested, what an independent review found, compatibility, and known limits.
- `CHANGELOG.md` — the full change record, with dated corrections.
- `docs/aws-deployment-guide.md` — deploying and operating VAREK Enterprise on AWS: architecture, IAM, keys, network, costs, sizing, health checks, backup and recovery, key rotation, upgrades and support, with an index to the AWS Foundational Technical Review requirements.
- `docs/security/threat-model.md` and `docs/security/threat-model-dataflow.md` — adversary models, in-scope guarantees, and out-of-scope non-goals.

- `docs/security/bypass-classes.md` — the bypass-class checklist and mediation-completeness argument, with per-class status.
- `docs/security/TRUSTED-COMPUTING-BASE.md` — per-component trusted-versus-verified status of the verification chain.
- `docs/security/v1.21-stage2-host-names.md` — the design for host names without agent DNS, with implementation notes and the review's findings.
- `docs/security/v1.25-wildcard-host-names.md` — the design for wildcard host names, with implementation notes and the decisions on budgets and the acknowledgment.
- `docs/security/v1.26-egress-proxy.md` — the design for the egress proxy, with implementation notes and the review's clarifications.
- `docs/security/v1.26.1-inspecting-mode.md` — the design for the inspecting mode, with implementation notes, the review's findings and the regression count.
- `docs/security/v1.27-program-launches.md` — the design for decided program launches, with implementation notes, the review's findings and the regression count.
- `docs/security/v1.27.1-writable-noexec.md` — the design for mounting writable directories `noexec`, with implementation notes, the review's findings and the regression count.
- `docs/security/v1.28-credential-release.md` — the design for credential release and revocation, with implementation notes, the review's findings and the regression count.
- `docs/security/v1.28.1-delegation-chain.md` — the design for the on-behalf-of chain (verifying the user and the agent, sub-agents by control group, expiry), with implementation notes, the review's findings and the regression count.
- `docs/security/v1.29-varek-run.md` — the design for `varek run` with nothing set up first (the baseline and profiles, opening as the agent, the dry run, `varek policy suggest`), with implementation notes and the regression count.
- `docs/security/v1.31-standards.md` — the design for MCP servers, MCP calls and tokens and the OCSF export, with implementation notes, the review's findings and the regression count.
- `docs/standards/agent-baseline.md` — each Agent Baseline control, what in VAREK addresses it and how far.
- `docs/security/v1.32-containers.md` — the design for the image and the container settings, with what each was measured to need, the review's findings and the results on each host; `packaging/container/README.md`, how to get, run and check the image.

- `docs/security/v1.34-kubernetes.md` — the design for the node agent, the NRI plugin, the launcher, policies as cluster resources and the evidence on the node, with the review's findings and the 24-hour runs; `packaging/helm/varek`, the Helm chart.
- `docs/security/v1.10-architecture-roadmap.md` — the model- and TCB-changing track.
- `docs/verification/` — the bitvector, bounded-string and bounded-sequence fragments, the verdict-distribution harness and its corpus schema.
- `docs/security/v1.9.2-baseline-allowlist.md` — the default-deny allowlist rationale and class-to-syscall map.
- `docs/adr/0001-syscall-layer.md` — the syscall-layer architecture decision.
- `v1_7/INTEGRATION-hotl.md` — using the progress verifier as an unattended-startup gate.
- `SECURITY.md` — supported versions, private vulnerability reporting, and the security contact (kenneth.douglas@soberagents.ai).

---

8 Roadmap — shrinking UNKNOWN without weakening soundness

The v1.10/v1.11 verification program has one goal: move cases out of UNKNOWN into a provable SATISFIED or UNSATISFIED, raising the clear rate on safe actions, under a soundness obligation that forbids ever turning an unsafe action into SATISFIED. Its first three releases have shipped:

1. **v1.13.0** — the SMT decision procedure in the enforcement path, the bitvector flag fragment, the prefix and equality part of the bounded string fragment, and the verdict-distribution harness with its `unsafe_satisfied == 0` gate.
2. **v1.14.0** — the rest of the bounded string fragment: suffix, contains and glob matchers, exact rule reachability with witnesses.
3. **v1.15.0** — certificates for every authorization, checked by an independent checker.

Next, stated as direction and not claimed as shipped:

- **v1.33.0** (planned as v1.30.0): macOS through a Linux VM. **v1.34.0** to **v1.36.0** (`docs/roadmap/v1.32-v1.35.md`): Kubernetes (a node agent that supervises agent pods from outside them, a Helm chart, OpenShift, an Amazon EKS add-on), agent frameworks, and security operations. After v1.31.0: an `env=` clause naming what a confined server's environment may hold; MCP tokens obtained by VAREK itself (token exchange) rather than from stores.

- **Bounded sequence fragment (v1.11, candidate).** Element-level reasoning for the cross-action data-flow subsystem, composed on top of the string and bitvector fragments so its soundness inherits theirs.
- **A customer baseline** for the verdict-distribution harness, on a design partner's workload and policy.
- **The v1.10 architecture track:** race-free filesystem enforcement and trusted-base reduction via a kernel-native sandbox (Landlock, used for launches from v1.27.0; `noexec` mounts for written files from v1.27.1); acquisition-time mediation tiering with a latency gate; an optional post-grant re-mediation mode; an escalation ladder for UNKNOWN at the kernel edge; and a trusted base reduced to a minimal supervisor, a small proof checker and the kernel.
- Expanded information-flow coverage including implicit flows; surface-language ergonomics for `var::`; and continued external audit and independent assurance engagement.

External validation context: the DARPA/NSF AI Forge program (June 2026) names provably secure-by-construction agent sandboxes with verifiable action and information-flow bounds and low-latency runtime intervention as a national priority — the problem class VAREK's shipped architecture addresses. This is cited as third-party validation of the problem, not as a claim of program involvement.

---

9 Licensing and intellectual property

VAREK is released under the **MIT license**. Three provisional patent applications are on file covering the formal-verification (SMT decision procedure) layer, the Warden kernel-level enforcement architecture, and action-graph compositional policy decision. The project is **patent-pending**; nothing in this release is granted. Non-provisional conversions begin in 2027. Relicensing considerations are deferred pending conversion.

A Appendix A — Version history

VERSION	FOCUS
v1.0 (Apr 2026)	Public launch. Formal-verification layer, MIT license.
v1.1	Security release: pluggable isolation backend; subprocess-boundary fix.
v1.4	The Warden: a seccomp user-notify supervisor in C (kernel-boundary enforcement).
v1.5	Fast-path matcher and SMT feasibility study for the decision layer.
v1.6	Plan-time verification of agent action-graphs; compositional three-state decision.
v1.7	Kernel-boundary integration; vertical stack from system boundary to formal decision.
v1.8	Cross-action data-flow verification; audited declassification (v1.8.0); bounded-refusal breaker (v1.8.2).
v1.9	Progress-safety verification. Load-time liveness proof; certified human-out-of-the-loop.
v1.9.1	Enforcement hardening. io_uring refused; TOCTOU-safe file-open mediation; connect and execve deny-only; threat model and trusted-computing-base published.
v1.9.2	Mediation completeness. Default-deny allowlist; native-ABI lockdown (x32 closed); hard-deny set; scalar-flag `CLONE_NEWUSER` denial; conformance target.
v1.9.3	Lifecycle coupling in the live Warden. PID namespace, death-signal coupling, fork-race guard, process-descriptor watch, crash test.
v1.12.0	Mediation correctness. Resolve-then-decide; escaped records carrying the resolved object; datagram egress refused; CycloneDX 1.6 evidence export.
v1.12.1	Side-effect-free denial (`O_PATH` pin, then open); authenticated verdict stream (run id, contiguous sequence); inbound networking refused;

VERSION	FOCUS
	agent in its own network namespace.
v1.12.2	Threads and child processes (`clone3` answered `ENOSYS`); launch exec granted once per run.
v1.12.3	Dynamically linked agents: symlinks followed and decided as their targets; `/proc/self` mapped to the agent.
v1.12.4	The `--plan` gate decides file opens on the lexically canonical declared path.
v1.13.0	SMT decision procedure in the enforcement path (first release of the v1.10 program). Bounded strings and bitvectors; read-only flag clauses; load-time rule reachability; solver cross-check; verdict-distribution harness (85.4% of safe opens cleared, 0 unsafe, on the seed corpus).
v1.14.0	Bounded string fragment: exact, suffix, contains and glob matchers on the resolved path; exact reachability with witnesses; derivative-based oracle.
v1.15.0	Certificates: every authorization checked by an independent checker before it takes effect; `tools/varek_audit.py`.
v1.16.0	Hash-chained, Ed25519-signed, anchored verdict stream; 4,096-token glob cap bounding every decision.
v1.16.1	Deployment preflight (`tools/varek_preflight.sh`); CI build with libseccomp and libsodium.
v1.16.2	Off-host anchor: forwarder and append-only receiver.
v1.16.3	The Verdict Service's plan checker in the repository; escaped JSON output.
v1.17.0	Protected objects refused by identity; raw devices refused; `stat`/`access`/`readlink` decided; agent runs unprivileged (`--run-as`, default `nobody`).
v1.18.0	The claims and the code agree. Plan gate runs the data-flow check, breaker and progress-safety proof; URL-host flow rules; signed, schema-tested CycloneDX export; `EACCES` in records; figures re-measured or restated.

VERSION	FOCUS
v1.19.0	A refusal limit per session (`session_refusal_budget`).
v1.20.0	Fields on plan steps; `trust_declared_fields`.
v1.21.0	Decided connections: each connect decided on the destination copied once, dialed by the Warden and handed to the agent; TCP, connected UDP, Unix sockets; tested with curl, Python and Node.js.
v1.21.1	Plan steps say how they open a file (`open=`), so the plan gate can authorize declared reads on read-only paths. Corrects the sample plan and the data-flow threat model.
v1.22.0	The `varek` command (`doctor`, `init`, `run`, `refusals`, `audit`, `export`, …) and `varek bench`, which measures the cost of mediation per call on the host it runs on and checks every verdict. Corrects the latency figure (Appendix B, item 8).
v1.23.0	VAREK Enterprise on AWS Marketplace: an Amazon Linux 2023 AMI with the Warden, the `varek` command, the Enterprise policy packs (HIPAA, SOC 2) and an AWS License Manager check. No decision changes.
v1.23.1	The Enterprise license check works with Marketplace contract entitlements; the Core packs read `/usr/share/`; the deployment guide for AWS. No decision changes.
v1.24.0	Host names without agent DNS: the Warden resolves every named host, serves the agent hosts and resolver views, refuses its DNS, and decides each connect on its address and every name of it; the audit checks each name against the lookup records. musl agents can open files.
v1.25.0	Wildcard host names, opt-in (`acknowledge=dns-channel`): shared domains refused at load, a stub resolver in the agent's namespace answering only names a rule allows, budgets on new names, every question recorded and audited, connects decided over any number of names on one address.
v1.26.0	The egress proxy, SNI mode (`proxy on`): connects on proxied ports decided on the TLS SNI or HTTP `Host` by an unprivileged proxy and the Warden, synthetic addresses for proxied names, plain HTTP checked

VERSION	FOCUS
	request by request, chaining to a customer's proxy (`proxy upstream`), every hand-off, decision and close recorded and audited.
v1.26.1	The egress proxy's inspecting mode (`proxy inspect`): `warden-proxy` as its own hashed binary, TLS terminated with a name-constrained CA made for the run, servers verified first, every request's method, path and query decided by request rules and certified before it is sent, `max_body`, passthrough hosts, every request and body recorded and audited.
v1.27.0	Decided program launches (`require warden 1.27`): `allow exec` rules expanded to a launch set held by Landlock, each launch decided on its name and certified, the identity check, launch records and their audit; what a compile needs.
v1.27.1	Nothing written during the run is mapped executable: under `require warden 1.27` the policy's writable directories mounted `noexec` where the Warden opens files, a descriptor not coming through such a mount refused, each directory recorded and audited against its rule; `--writable-exec` to turn it off for a run.
v1.28.0	Credential release: the agent holds only placeholders; `warden-secrets` fetches from a file, Vault or AWS Secrets Manager and hands the values to the proxy, which writes one into a request only in the header an allowing rule names; `varek revoke`; each credential recorded and every request that carried one audited against its rule.
v1.28.1	The on-behalf-of chain in every decision record: the user's token verified against a trusted identity provider's keys, the agent's X.509-SVID against its trust domain's bundle, each sub-agent attributed by a control group of its own from its launch; the token's expiry revokes the run; the chain recorded and audited, not enforced.
v1.29.0	`varek run` with nothing set up first: a baseline policy and agent profiles; files opened and changed by name as the agent's own user; a dry run that records what a policy refuses and enforces only a fixed set; `varek policy suggest`.
v1.31.0	MCP servers confined to their own rules (`mcp server`, `for=`), launched only with an environment that names no code; MCP calls over HTTP decided by method and tool; MCP tokens bound to their server and the

VERSION	FOCUS
	agent's own checked; OCSF 1.9.0 events with `--verify` and `--follow`; the Agent Baseline mapping; the rule cap 512.
`v1.31.1`	A patch release: `epoll_wait`, `epoll_create`, `pipe` and `getpgrp` admitted under every policy (asyncio, a shell's command substitution, bash).
`v1.32.0`	Containers: an official, signed image with its SBOM; Docker and Podman, rootless included, under ten capabilities and the runtime's seccomp profile, never `--privileged`; AppArmor and SELinux profiles for the Warden's mounts; sub-agents and confined MCP servers in a container; `varek doctor` and `varek run` checking the container.
`v1.32.1`	A patch release: rootless Podman runs sub-agents and confined MCP servers without `--systemd=always`.
`v1.34.0`	**Kubernetes: a node agent (one Warden per node, the runtime handing it each container's seccomp notifications), an NRI plugin, the `varek-init` launcher, `VarekPolicy` resources and an admission webhook, per-container streams exported as OCSF, and a signed Helm chart; the agent's pod passes Pod Security's "restricted" level; 24-hour runs passed on Amazon EKS and OpenShift.**
(planned)	macOS through a Linux VM (planned as v1.30.0, then v1.33.0; takes the next number when it is released).
`v1.35.0 (planned)`	Agent frameworks: plans generated from the framework, Amazon Bedrock AgentCore first.
`v1.36.0 (planned)`	Security operations: Splunk, Microsoft Sentinel, OpenTelemetry.
`v1.11 (candidate)`	Bounded sequence fragment for cross-action data flow.

B Appendix B — Corrections to earlier editions

Earlier editions of this paper (v1.9.1 through v1.31.0) made statements a later review of the code did not bear out. They are corrected in the text above; this list records what changed.

1. **UNKNOWN diagnostics and resource bounds.** Specified for v1.9.1 but not implemented then. §3.2 describes what exists, from v1.13.0. (Corrected in v1.18.0.)

2. **TOCTOU figure.** "510 leaks in 20,000 attempts" has no recorded run. The v1.18.0 re-run gave 1,848 to 1,889 leaks for approve-then-continue and 0 for resolve-and-inject. (Corrected in v1.18.0.)
3. **io_uring test.** `v1_7/tests/test_v191_io_uring.c` does not check io_uring under the Warden filter on its own; `test_v14_filter` does, from v1.18.0. (Corrected in v1.18.0.)
4. **Data-flow, breaker and progress-safety in the Warden.** Through v1.17.0 these were a library with tests that the Warden did not call; the `--plan` gate runs them with `--flow-policy` from v1.18.0. (Corrected in v1.18.0.)
5. **Connect deny-only.** True from v1.9.1 through v1.20.0; from v1.21.0 connects are decided and dialed by the Warden. (Corrected in v1.21.0 and v1.21.1.)
6. **`/proc/self`.** It is an ordinary symlink, not a magic link, and disabling magic-link resolution does not refuse it. The no-symlinks flag refused it in v1.12.0 to v1.12.2; from v1.12.3 a leading `/proc/self` is mapped to the agent and any other process's `/proc/<pid>` is refused after resolution. (Corrected in v1.21.0.)
7. **Benchmark figures.** A v1.4 P99 of 57 µs is 44 µs in the recorded run; "zero false negatives" is a count of 0 in one benchmark, not a proof. (Corrected in v1.18.0.)
8. **Decision latency.** The v1.21.1 edition gave the median across all decisions as 8 µs "including the notification round trip". That figure, from the v1.14 to v1.16 release notes, is the Warden's own time (its clock starts when the notification is received, so the round trip is not in it), over the v1.4 `bench_target` mix, in which every connect was refused with nothing dialed. §2 now gives `varek bench` figures per kind of call, both the Warden's time and what the agent waits. (Corrected in v1.22.0.)
9. **Where the `noexec` mounts go.** The v1.27.0 edition said v1.27.1 would mount the writable paths `noexec` in the agent's own mount namespace. That would not have worked: the Warden opens every file for the agent, and the kernel checks the mount the file was opened through. They are in the Warden's own namespace, which the agent inherits (§2.23). (Corrected in v1.27.1.)

Copyright Sober Agentic Infrastructure, Inc. VAREK is open source under the MIT license.